

Deckhouse Kubernetes Platform

Справочник администратора

Версия v1.76

Дата генерации: 15.09.2026

Содержание

Содержание	2
Документация Deckhouse Kubernetes Platform	73
Установка	73
Способы установки	73
Варианты установки	73
Требования к установке	73
Подготовка конфигурации	74
Файл конфигурации установки	74
Post-bootstrap-скрипт	79
Установка	79
Проверки перед началом установки	80
Откат установки	82
Закрытое окружение, работа через прокси-сервер и стороннее хранилище образов контейнеров	82
Установка из стороннего хранилища образов контейнеров	82
Особенности настройки Nexus	84
Особенности настройки Harbor	89
Ручная загрузка образов DKP и БД уязвимостей в приватное хранилище образов контейнеров	90
Создание кластера и запуск DKP без использования каналов обновлений	93
Использование прокси-сервера	93
Автозагрузка прокси-переменных пользователям в CLI	93
Дополнительные ресурсы	94
Как настроить?	94
Основы конфигурации Deckhouse	94
Изменение конфигурации кластера	95
Изменение защищённых параметров	95
Изменение защищённых параметров с использованием dhctl	96
Изменение защищённых параметров вручную	96
Просмотр текущих настроек	97
Настройка модуля	97
Включение и отключение модуля	98
Наборы модулей	98
Особенности работы с набором модулей Minimal	99
Доступ к документации текущей версии	100
Управление размещением компонентов Deckhouse	100
Выделение узлов под определенный вид нагрузки	100
Особенности автоматки, зависящие от типа модуля	100
Дополнительные ресурсы	101
Работа с container registries и редакциями	101
Работа с registry	101
Работа с редакциями	101
Переключение работающего кластера DKP на использование стороннего registry	101
Дополнительные ресурсы	102
Восстановление подключения к registry при истекшем и неверном лицензионном токене	102
Переключение редакций	103
Переключение DKP с EE на CSE	103
Переключение с использованием модуля registry	103
Переключение без модуля registry	106
Переключение DKP на CE/BE/SE/SE+/EE	111
Переключение с помощью модуля registry	111
Переключение без использования модуля registry	115
Дополнительные ресурсы	118
Аутентификация	118
Дополнительные ресурсы	118
Интеграция с внешними провайдерами аутентификации	118
Общая схема интеграции	119
Интеграция по OIDC (OpenID Connect)	120
Keycloak	120
Blitz Identity Provider	121

Okta	121
Интеграция по SAML 2.0	122
Интеграция по LDAP	122
Интеграция с GitHub	123
Интеграция с GitLab	123
Интеграция с Atlassian Crowd	124
Интеграция с Bitbucket Cloud	124
Дополнительные ресурсы	125
Локальная аутентификация	125
Создание статического пользователя	125
Операции над локальным пользователем	125
Административные операции	125
Сброс пароля пользователем	126
Добавление пользователя в группу	126
Настройка парольной политики	127
Настройка двухфакторной аутентификации (2FA)	127
Дополнительные ресурсы	127
Доступ к Kubernetes API через балансировщик трафика	127
Как работает защита доступа к Kubernetes API	128
Доступ с использованием базовой аутентификации (LDAP)	128
SSO по Kerberos (SPNEGO) для LDAP	129
Дополнительные ресурсы	130
Настройка ресурсов	130
Пример конфигурации	130
Дополнительные ресурсы	131
Авторизация	131
Кому и когда выдаются права	131
Аутентификация пользователей	132
Аутентификация сервисных аккаунтов	132
Дополнительные ресурсы	132
Текущая модель авторизации	132
Высокоуровневые роли, используемые для реализации модели	132
Список доступа для каждой высокоуровневой роли по умолчанию	133
Пример AuthorizationRule	134
Пример ClusterAuthorizationRule	135
Расширение прав доступа для высокоуровневых ролей	135
Дополнительные ресурсы	136
Экспериментальная модель авторизации	136
Use-роли	137
Manage-роли	137
Подсистемы ролевой модели	138
Состав подсистем ролевой модели	138
Создание новой роли подсистемы	139
Описание лейблов	140
Описание селекторов агрегации	140
Расширение пользовательской роли	140
Расширение существующих manage-subsystem-ролей	141
Расширение manage-subsystem-ролей с добавлением нового пространства имён	141
Расширение существующих use-ролей	142
Дополнительные ресурсы	142
Модель административного доступа к кластеру	142
Файлы kubeconfig на master-узлах	142
Административный доступ на основе RBAC	143
Рекомендуемый административный доступ	143
Символическая ссылка root kubeconfig	143
Усиление безопасности	143
Аварийный доступ (break-glass) к кластеру	143
Дополнительные ресурсы	143
Выдача прав пользователям и сервисным аккаунтам	144
Предоставление прав с помощью AuthorizationRule и ClusterAuthorizationRule (текущая ролевая модель)	144
Предоставление прав пользователю в рамках одного пространства имен	144

Предоставление прав пользователю во всех пространствах имен	144
Предоставление прав с помощью ClusterRoleBinding и RoleBinding (экспериментальная ролевая модель)	145
Назначение прав администратору кластера (экспериментальная ролевая модель)	145
Назначение прав сетевому администратору (экспериментальная ролевая модель)	145
Назначение административных прав пользователю в рамках пространства имён (экспериментальная ролевая модель)	147
Дополнительные ресурсы	148
Проверка наличия прав у пользователя	148
Дополнительные ресурсы	149
Доступ для CI/CD	149
ServiceAccount	149
Предварительные требования	149
Создание ServiceAccount и долгоживущего токена	149
Выдача прав	150
Получение URL API и сертификата CA	150
Создание kubeconfig	151
Короткоживущие токены (TokenRequest)	151
Типовые сценарии к использованию в CI	151
Basic Auth	151
Предварительные требования	152
Включение	152
Получение эндпоинта API	152
Проверка	152
Настройка kubeconfig	152
Использование в GitLab CI	152
Обмен токена (token exchange)	152
Предварительные требования	153
Создание DexClient	153
Получение URL Dex и API	153
Выдача RBAC	153
Запрос обмена токена	154
GitLab CI	154
Keycloak	155
Устранение неполадок	155
Дополнительные ресурсы	156
Интеграция с IaaS	156
Публичные облачные провайдеры	156
Подключение и авторизация в Yandex Cloud	156
Подготовка окружения	156
Создание сервисного аккаунта	157
Назначение IAM-ролей	157
Генерация авторизованного ключа	157
Проверка и увеличение квот	157
Резервирование публичного IP	157
Дополнительные ресурсы	158
Схемы размещения и настройка Yandex Cloud	158
Схемы размещения	158
Standard	158
WithoutNAT	159
WithNATInstance	161
Назначение YandexClusterConfiguration	163
Пример конфигурации	164
Сетевые параметры и безопасность	164
Внутренняя адресация узлов кластера	164
Назначение внешних IP-адресов и исходящего трафика	165
Настройка DNS и DHCP для внутренних сетей	165
Использование заранее созданных подсетей	165
Дополнительные внешние сети	166
Настройка групп безопасности в Yandex Cloud	166
Настройка доступа через bastion-хост	167
Дополнительные ресурсы	168

Хранилище и балансировка нагрузки в Yandex Cloud	168
Хранилище (CSI и StorageClass)	168
Исключение ненужных StorageClass	168
Назначение StorageClass по умолчанию	169
Изменение размера PVC	169
Балансировка нагрузки	171
Внешний LoadBalancer	171
Внутренний LoadBalancer	171
Аннотации объекта Service	172
Использование отдельной целевой группы для NodeGroup	172
Проверки состояния целевой группы	173
Особенности применения изменений	173
Изменение networkType не пересоздаёт CloudEphemeral-узлы	173
Интеграция вручную созданных VM	174
Дополнительные ресурсы	174
Интеграция со службами Yandex Cloud	174
Интеграция с Yandex Lockbox	174
Интеграция с Yandex Managed Service for Prometheus	176
Дополнительные ресурсы	177
Подключение и авторизация в VK Cloud	177
Схемы размещения и настройка VK Cloud	177
Схемы размещения	177
Standard	177
StandardWithNoRouter	179
Simple	181
SimpleWithInternalNetwork	183
Конфигурация	185
Примеры конфигурации	186
Пример 1	186
Пример 2	186
Список необходимых сервисов	186
Настройка LoadBalancer	187
Пример IngressNginxController	187
Настройка и политики безопасности на узлах кластера	187
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	187
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	188
Как загрузить образ в VK Cloud	188
Проверка поддержки провайдером группы безопасности (security groups)	189
Настройка работы онлайн-изменений размера дисков	189
Использование rootDiskSize	190
Диски в VK Cloud	190
Параметр rootDiskSize	190
Рекомендация для master-узлов и бастiona — сетевой диск	190
Рекомендация для ephemeral-узлов — локальный диск	190
Как проверить объем диска в flavor	190
Как переопределить тип диска по умолчанию облачного провайдера	191
Подключение и авторизация в Selectel Cloud	191
Схемы размещения и настройка Selectel Cloud	191
Схемы размещения	191
Standard	191
StandardWithNoRouter	193
Simple	195
SimpleWithInternalNetwork	197
Конфигурация	199
Примеры конфигурации	200
Пример 1	200
Пример 2	200
Список необходимых сервисов	200
Настройка LoadBalancer	201
Пример IngressNginxController	201
Настройка и политики безопасности на узлах кластера	201

Установка дополнительных групп безопасности (security groups) на статических и master-узлах	201
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	202
Как загрузить образ в Selectel	202
Проверка поддержки провайдером группы безопасности (security groups)	203
Настройка работы онлайн-изменений размера дисков	203
Использование rootDiskSize	204
Диски в Selectel	204
Параметр rootDiskSize	204
Рекомендация для master-узлов и бастiona — сетевой диск	204
Рекомендация для ephemeral-узлов — локальный диск	204
Как проверить объем диска в flavor	204
Как переопределить тип диска по умолчанию облачного провайдера	205
Оффлайн-изменение размера диска	205
Подключение и авторизация в OVH Cloud	205
Схемы размещения и настройка OVH Cloud	205
Схемы размещения	205
Standard	205
StandardWithNoRouter	207
Simple	209
SimpleWithInternalNetwork	211
Конфигурация	213
Примеры конфигурации	214
Пример 1	214
Пример 2	214
Список необходимых сервисов	214
Настройка LoadBalancer	215
Пример IngressNginxController	215
Настройка и политики безопасности на узлах кластера	215
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	215
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	216
Как загрузить образ в OVHcloud	216
Проверка поддержки провайдером группы безопасности (security groups)	217
Настройка работы онлайн-изменений размера дисков	217
Использование rootDiskSize	218
Диски в OVHcloud	218
Параметр rootDiskSize	218
Рекомендация для master-узлов и бастiona — сетевой диск	218
Рекомендация для ephemeral-узлов — локальный диск	218
Как проверить объем диска в flavor	218
Как переопределить тип диска по умолчанию облачного провайдера	219
Оффлайн-изменение размера диска	219
Подключение и авторизация в OpenStack	219
Схемы размещения и настройка OpenStack	220
Схемы размещения	220
Standard	220
StandardWithNoRouter	221
Simple	223
SimpleWithInternalNetwork	225
Конфигурация	227
Примеры конфигурации	228
Пример 1	228
Пример 2	228
Список необходимых сервисов	228
Настройка LoadBalancer	229
Пример IngressNginxController	229
Настройка и политики безопасности на узлах кластера	229
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	229
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	230
Как загрузить образ в OpenStack	230

Проверка поддержки провайдером группы безопасности (security groups)	231
Настройка работы онлайн-изменений размера дисков	231
Использование rootDiskSize	232
Диски в OpenStack	232
Параметр rootDiskSize	232
Рекомендация для master-узлов и бастiona — сетевой диск	232
Рекомендация для ephemeral-узлов — локальный диск	232
Как проверить объем диска в flavor	232
Как переопределить тип диска по умолчанию облачного провайдера	233
Оффлайн-изменение размера диска	233
Подключение и авторизация в Amazon Web Services	233
Требования	233
Доступ к AWS API	233
Настройка IAM через веб-интерфейс AWS	235
Настройка IAM через AWS CLI	236
Настройка IAM через Terraform	237
Использование своей IAM-роли для узлов (iamNodeRole)	237
Дополнительные ресурсы	238
Схемы размещения и настройка Amazon Web Services	238
WithoutNAT	238
WithNAT	239
Назначение AWSClusterConfiguration	241
Внутренняя адресация и подсети	242
Группы безопасности	242
Настройка пирингового соединения между VPC	242
Настройка доступа через bastion-хост	242
Создание кластера в новом VPC с доступом через имеющийся bastion-хост	243
Создание кластера в новом VPC и развертывание bastion-хоста для доступа к узлам	243
Использование существующего VPC (existingVPCID)	243
Дополнительные ресурсы	243
Хранилище и балансировка нагрузки в Amazon Web Services	243
Хранилище (CSI и StorageClass)	243
Увеличение объема тома (volume)	244
Балансировка нагрузки	244
Настройка балансировщика при отсутствии Ingress-узлов в некоторых зонах	244
Подключение CloudStatic-узлов	245
Дополнительные ресурсы	245
Интеграция со службами Amazon Web Services	245
Работа с Amazon Elastic Container Registry (ECR)	245
Дополнительные ресурсы	246
Подключение и авторизация в Microsoft Azure	246
Требования	246
Доступ к Azure API	246
Дополнительные ресурсы	246
Схемы размещения и настройка Microsoft Azure	247
Standard	247
Хранилище и балансировка нагрузки в Microsoft Azure	247
Хранилище	247
Балансировка нагрузки	248
Дополнительные ресурсы	248
Интеграция со службами Microsoft Azure	248
Поддержка Service Endpoints	248
Дополнительные ресурсы	249
Подключение и авторизация в Google Cloud Platform	249
Создание Service Account	249
Настройка через Google Cloud Console	249
Настройка через Google Cloud CLI	249
Использование созданного Service Account	250
Дополнительные ресурсы	250
Схемы размещения и настройка Google Cloud Platform	250
Схемы размещения	250
Standard	250

WithoutNAT	252
Конфигурация	254
Включение вложенной виртуализации	255
Добавление дополнительных дисков	255
Настройка политик безопасности на узлах	256
Установка дополнительных network tags на статических и master-узлах	256
Установка дополнительных network tags на эфемерных узлах	256
Добавление CloudStatic узлов в кластер	256
Дополнительные ресурсы	256
Частные облака	256
Подключение и авторизация в VK Cloud	256
Схемы размещения и настройка VK Cloud	257
Схемы размещения	257
Standard	257
StandardWithNoRouter	259
Simple	260
SimpleWithInternalNetwork	262
Конфигурация	264
Примеры конфигурации	265
Пример 1	265
Пример 2	265
Список необходимых сервисов	265
Настройка LoadBalancer	266
Пример IngressNginxController	266
Настройка и политики безопасности на узлах кластера	266
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	266
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	267
Как загрузить образ в VK Private Cloud	267
Проверка поддержки провайдером группы безопасности (security groups)	268
Настройка работы онлайн-изменений размера дисков	268
Использование rootDiskSize	269
Диски в VK Private Cloud	269
Параметр rootDiskSize	269
Рекомендация для master-узлов и бастiona — сетевой диск	269
Рекомендация для ephemeral-узлов — локальный диск	269
Как проверить объем диска в flavor	269
Как переопределить тип диска по умолчанию облачного провайдера	270
Подключение и авторизация в OpenStack	270
Схемы размещения и настройка OpenStack	270
Схемы размещения	270
Standard	270
StandardWithNoRouter	272
Simple	274
SimpleWithInternalNetwork	276
Конфигурация	278
Примеры конфигурации	279
Пример 1	279
Пример 2	279
Список необходимых сервисов	279
Настройка LoadBalancer	280
Пример IngressNginxController	280
Настройка и политики безопасности на узлах кластера	280
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	280
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	281
Как загрузить образ в OpenStack	281
Проверка поддержки провайдером группы безопасности (security groups)	282
Настройка работы онлайн-изменений размера дисков	282
Использование rootDiskSize	283
Диски в OpenStack	283
Параметр rootDiskSize	283

Рекомендация для master-узлов и бастiona — сетевой диск	283
Рекомендация для ephemeral-узлов — локальный диск	283
Как проверить объем диска в flavor	283
Как переопределить тип диска по умолчанию облачного провайдера	284
Оффлайн-изменение размера диска	284
Подключение и авторизация в Huawei Cloud	284
Требования	284
Доступ к Huawei Cloud API	284
Создание группы пользователей	284
Создание IAM-пользователя	284
JSON-политики	285
Схемы размещения и настройка Huawei Cloud	287
Standard	287
VpcPeering	288
Хранилище и балансировка нагрузки в Huawei Cloud	289
Хранилище	289
Балансировка нагрузки	289
Дополнительные ресурсы	289
Интеграция со службами Huawei Cloud	289
Работа с InstanceClass	289
Дополнительные ресурсы	290
Подключение и авторизация в Basis Dynamix	290
Требования	290
Подготовка cloud-образа	290
Настройка доступа к portalу	290
Дополнительные ресурсы	291
Схемы размещения и настройка Basis Dynamix	291
Standard	291
StandardWithInternalNetwork	292
Обязательные параметры	292
Хранилище и балансировка нагрузки в Basis Dynamix	292
Хранилище	292
Балансировка нагрузки	293
Настройка Service типа LoadBalancer	293
Дополнительные ресурсы	293
Интеграция с облаком Basis Dynamix	293
Основные возможности	293
Рекомендации	294
Дополнительные ресурсы	294
Системы виртуализации	294
Интеграция с облаком Deckhouse Virtualization Platform	294
Дополнительные ресурсы	294
Подключение и авторизация в Deckhouse Virtualization Platform	294
Создание пользователя	295
Добавление роли	295
Генерация kubeconfig	295
Схемы размещения и настройка Deckhouse Virtualization Platform	296
Конфигурация	297
Дополнительные ресурсы	298
Подключение и авторизация в VMware Cloud Director	299
Подготовка ресурсов	299
Права пользователя	299
Добавление сети	302
Настройка DHCP	305
Добавление vApp	307
Добавление сети к vApp	308
Входящий трафик	310
Настройка правил DNAT/SNAT на edge gateway	310
Выпуск сертификатов Let's Encrypt	311
Настройка firewall	311
Шаблон виртуальной машины	313
Подготовка шаблона из OVA-файла	314

Настройка шаблона в VCD	319
Использование хранилища	324
Использование балансировщика нагрузки	324
Дополнительные ресурсы	324
Схемы размещения и настройка VMware Cloud Director	325
Схемы размещения	325
Standard	325
WithNAT	325
Конфигурация	328
Пример конфигурации кастомного ресурса VCDInstanceClass	329
Storage	329
CSI	329
Дополнительные ресурсы	329
Подключение и авторизация в VMware vSphere	329
Требования	329
Требования к образу виртуальной машины	330
Подготовка образа виртуальной машины	330
Конфигурация vSphere	331
Настройка через vSphere Client	331
Создание тегов и категорий тегов с использованием vSphere Client	331
Настройка Datastore с использованием vSphere Client	334
Создание и назначение роли с использованием vSphere Client	335
Настройка через govc	336
Установка govc	336
Создание тегов и категорий тегов с использованием govc	336
Настройка Datastore с использованием govc	337
Создание и назначение роли с использованием govc	337
Дополнительные ресурсы	338
Схемы размещения и настройка VMware vSphere	338
Standard	338
Список необходимых привилегий	340
Дополнительные ресурсы	343
Хранилище и балансировка нагрузки в VMware vSphere	343
Хранилище	343
Изменение размера тома (PVC)	343
Балансировка нагрузки	343
CSI	344
Важная информация об увеличении размера PVC	344
Настройка Datastore	344
Дополнительные ресурсы	344
Интеграция со службами VMware vSphere	344
Управление ресурсами vSphere	344
Удаление CloudPermanent-узлов в vSphere	344
Уменьшение количества узлов	345
Полное удаление группы узлов	345
Дополнительные ресурсы	346
Подключение и авторизация в zVirt	346
Требования	346
Подготовка cloud-образа	346
Подготовка шаблона виртуальной машины	347
Схемы размещения и настройка zVirt	349
Standard	350
Дополнительные ресурсы	351
Хранилище и балансировка нагрузки в zVirt	351
Хранилище	351
Требования	351
Конфигурация	351
Балансировка нагрузки	351
Дополнительные ресурсы	351
Интеграция со службами zVirt	351
Основные возможности	352
Дополнительные ресурсы	352

Гибридные кластеры	352
Типы групп узлов	352
Способы добавления узлов	352
Общие сетевые требования	352
Дополнительные ресурсы	353
Гибридный кластер с Yandex Cloud	353
Предварительные требования	353
Добавление автоматически создаваемых узлов	353
Добавление вручную созданных узлов через bootstrap-скрипт	357
Дополнительные ресурсы	359
Гибридный кластер с VCD	359
Предварительные требования для VCD	359
Добавление автоматически создаваемых узлов	359
Добавление вручную созданных узлов через bootstrap-скрипт	360
Дополнительные ресурсы	362
Гибридный кластер с vSphere	362
Предварительные требования для vSphere	362
Добавление автоматически создаваемых узлов	362
Добавление вручную созданных узлов через bootstrap-скрипт	364
Дополнительные ресурсы	366
Гибридный кластер с DVP	366
Предварительные требования	366
Добавление автоматически создаваемых узлов	367
Добавление вручную созданных узлов через bootstrap-скрипт	369
Дополнительные ресурсы	370
Высокая надежность и доступность	370
Дополнительные ресурсы	370
Рекомендации по конфигурации узлов кластера и предотвращению перегрузки	370
Master-узлы	370
Frontend-узлы	371
Узлы мониторинга	371
Системные узлы	371
Предотвращение перегрузки Kubernetes API (FlowSchema)	371
Задержка перезагрузки или выключения узла, пока на нём работают критичные поды	371
Включение задержки выключения узла	371
Поиск подов, блокирующих выключение, и диагностика на узле	372
Дополнительные ресурсы	372
Управление режимом HA	372
Глобальное включение режима HA	372
Через кастомный ресурс ModuleConfig/global	372
Через веб-интерфейс Deckhouse	373
Настройка режима HA с двумя master-узлами и arbiter-узлом	373
Настройка в облачном кластере	373
Настройка в статическом кластере	374
Включение режима HA для отдельных компонентов	375
Дополнительные ресурсы	375
Хаос-инжиниринг	375
Дополнительные ресурсы	376
Масштабирование платформы и управление узлами	376
Общее управление и конфигурация control plane	376
Основные возможности	376
Включение, отключение и настройка модуля	376
Включение / отключение	376
Настройка	377
Проверка состояния и очередей DKP	377
Управление сертификатами	378
Управление PKI	378
Дополнительные SAN и автоматическое обновление	378
Ротация сертификатов kubelet	378
Ручное обновление сертификатов компонентов control plane	378
Ускорение перезапуска подов при потере связи с узлом	379
Принудительное отключение IPv6 на узлах кластера	379

Дополнительные ресурсы	380
Восстановление и отладка control plane	380
Восстановление при ошибках	380
Восстановление работоспособности кластера etcd	380
Восстановление master-узла при ошибке загрузки компонентов control plane через kubelet	380
containerd	380
Восстановление etcd	380
Просмотр списка узлов кластера в etcd	380
Способ 1	380
Способ 2	381
Восстановление кластера etcd при полной недоступности	381
Восстановление etcd при ошибке panic: unexpected removal of unknown remote peer	381
Действия при переполнении базы данных etcd (превышение quota-backend-bytes)	381
Дефрагментация etcd	382
Как выполнить дефрагментацию etcd узла в кластере с одним master-узлом	383
Как выполнить дефрагментацию etcd в кластере с несколькими master-узлами	383
Отказоустойчивость	384
Дополнительные ресурсы	385
Масштабирование и изменение master-узлов	385
Масштабирование и переход single-master/multi-master	385
Режимы работы control plane	385
Автоматическое масштабирование master-узлов	385
Типовые сценарии масштабирования	385
Удаление роли master с узла без удаления самого узла	386
Изменение образа ОС master-узлов в мультимастерном кластере	387
В облачном кластере	387
В статическом кластере	388
Изменение образа ОС в кластере с одним master-узлом	389
В облачном кластере	390
В статическом кластере	390
Добавление master-узлов в статический или гибридный кластер	390
Добавление master-узлов в облачном кластере	391
Уменьшение числа master-узлов в облачном кластере	392
Доступ к контроллеру DKP в мультимастерном кластере	393
Дополнительные ресурсы	393
Основы управления узлами в Deckhouse	393
Автоматическое развёртывание и обновление	393
Базовая настройка узлов и операционной системы	394
Обновления, требующие прерывания работы узла	394
Пример системной NodeGroup	395
Примеры описания NodeGroupConfiguration	395
Установка плагина cert-manager для kubectl на master-узлах	395
Конфигурация werf для игнорирования состояния Ready у группы узлов	395
Настройки для групп с узлами Static и CloudStatic	395
Запуск DKP на произвольном узле	396
Дополнительные ресурсы	396
Добавление и управление облачными узлами	397
Добавление CloudEphemeral-узлов в облачном кластере	397
Настройки для групп с узлами CloudEphemeral	398
Изменение конфигурации облачного провайдера в кластере	398
Автомасштабирование группы узлов	398
Пример сценария автомасштабирования	399
Выделение узлов под специфические нагрузки	399
Ускорение заказа узлов в облаке при горизонтальном масштабировании приложений	400
Пример описания NodeGroup	400
Облачные узлы	400
Конфигурация CloudEphemeral-узлов через NodeGroupConfiguration	401
Примеры описания NodeGroupConfiguration	401
Задание параметра sysctl	401
Добавление корневого сертификата в хост	401
Добавление сертификата в ОС и containerd	402

Обновление ядра на узлах	404
Для дистрибутивов, основанных на Debian	404
Для дистрибутивов, основанных на CentOS	404
Добавление CloudPermanent-узлов в облачном кластере	405
Добавление CloudStatic узла в кластер	406
Вручную	406
С помощью Cluster API Provider Static	407
С помощью Cluster API Provider Static для нескольких групп узлов	409
Добавление master-узлов в облачном кластере	410
Использование NodeGroup с приоритетом	411
Состояния группы узлов и их интерпретация	411
Дополнительные ресурсы	413
Добавление и управление bare-metal узлами	413
Добавление узлов в bare-metal кластере	413
Ручной способ	413
Автоматический способ	414
Изменение конфигурации статического кластера	415
Перемещение статического узла между NodeGroup	415
Ручная очистка статического узла	416
Пример описания NodeGroup	416
Пример описания NodeGroup для статических узлов	416
Изменение CRI для NodeGroup	416
Изменение NodeGroup у статического узла	417
Изменение IP-адреса в StaticInstance	417
Удаление StaticInstance	417
Дополнительные ресурсы	418
Пользовательские настройки на узлах	418
Мониторинг выполнения скриптов	420
Механизм повторного запуска скриптов	421
Особенности написания скриптов	421
Как использовать containerd с поддержкой Nvidia GPU	421
Ubuntu	422
CentOS	423
Особенности использования MIG в кластере из одного узла	424
Проверка успешности установки	425
Как развернуть кастомный конфигурационный файл containerd	426
Добавление конфигурации для дополнительного registry	428
Устаревший способ добавления конфигурации для дополнительного registry	429
Добавление авторизации в дополнительный registry (устаревший способ)	429
Настройка сертификата для дополнительного registry (устаревший способ)	430
Добавление TLS skip verify (устаревший способ)	431
Настройка зеркала для доступа к публичным registries (устаревший способ)	432
Новый способ добавления конфигурации для дополнительного registry	433
Добавление авторизации в дополнительный registry (актуальный способ)	434
Настройка сертификата для дополнительного registry (актуальный способ)	435
Как добавить TLS skip verify (актуальный способ)?	435
Настройка зеркала для доступа к публичным registries (актуальный способ)	436
Как автоматически проставить на узел кастомные лейблы	437
Дополнительные ресурсы	437
Миграция container runtime на containerd v2	438
Требования	438
Как включить containerd v2	438
Дополнительные ресурсы	439
Масштабирование приложений и управление ресурсами	439
Масштабирование по метрикам	440
Масштабирование по метрикам	440
Настройка	440
Настройка метрик через ресурсы модуля observability	441
Работа с нестабильными метриками	442
Получение значений метрик	442
Настройка автомасштабирования в веб-интерфейсе Deckhouse	442
Дополнительные ресурсы	443

Горизонтальное масштабирование	443
Как работает горизонтальное масштабирование (HPA)	443
Доступные типы метрик для HPA	443
Рекомендации HPA	444
Ограничения HPA	444
Как включить или отключить HPA	444
Настройка HPA	444
Примеры настройки HPA	445
Масштабирование по CPU и памяти	445
Масштабирование по метрикам объектов	446
Масштабирование на основе внешних данных	447
Дополнительные ресурсы	448
Вертикальное масштабирование	448
Как работает вертикальное масштабирование (VPA)	448
Как включить или отключить VPA	448
Настройка VPA	449
Пример настройки модуля	449
Дополнительные ресурсы	450
Перезапуск подов при изменении конфигурации	450
Поддерживаемые аннотации	450
Примеры использования	451
Слежение за всеми изменениями во всех подключенных ресурсах	451
Слежение за изменениями только в конкретных ресурсах	452
Слежение за изменениями в ресурсах из списка	453
Как включить или отключить перезапуск подов	454
Настройка	454
Дополнительные ресурсы	455
Планировщик	455
Распределение подов	455
Общее устройство планировщика	455
Этапы планирования подов	455
Как изменить или расширить логику работы планировщика	456
Дополнительные механизмы управления размещением подов	457
Профили планировщика	458
Перераспределение подов	458
Учет класса приоритета пода	459
Как DKP учитывает класс приоритета	459
Какие поды не вытесняются	459
Как включить или отключить перераспределение	459
Стратегии перераспределения	460
Дополнительные ресурсы	463
Классы приоритета	463
Доступные классы приоритета	463
Создание дополнительных классов приоритета	464
Механизм вытеснения	465
Принцип работы	465
Защита пода от вытеснения	465
Отличия классов приоритета от других механизмов управления ресурсами	466
Эксплуатация и диагностика	466
Дополнительные ресурсы	466
Сеть	466
Настройка и управление входящим трафиком	467
Управление исходящим трафиком	467
Настройка внутренней сети	467
Настройка межкластерного взаимодействия	467
Сетевые политики	467
Дополнительные ресурсы	467
Настройка внутренней сети	467
Режимы работы внутренней сети	467
Настройка режима	468
Сервисы	468
Поддерживаемые типы сервисов	468

Настройка сети сервисов через kube-proxy	469
Дополнительные ресурсы	469
Шифрование трафика между подами	469
Пример настройки mTLS	469
Дополнительные ресурсы	473
Настройка визуализации сетевого стека кластера	474
Дополнительные ресурсы	474
Переключение CNI с Flannel или Simple bridge на Cilium	474
Балансировка входящего трафика	476
Балансировка на сетевом уровне (NLB)	476
Балансировка на прикладном уровне (ALB)	476
Отличие Kubernetes Gateway API от API Gateway	476
Разделение ролей в модели Gateway API	476
Как выбрать реализацию ALB	476
Сравнение возможностей модулей ingress-nginx и alb	477
Следующие шаги	478
Дополнительные ресурсы	478
Балансировка средствами MetalLB	478
Режим Layer 2	478
Принцип работы	479
Преимущества модуля перед классическим MetalLB	479
Сравнение поведения	479
Пример использования MetalLB в режиме L2 LoadBalancer	479
Режим BGP	481
Принцип работы	481
Конфигурация	481
Установка BGP-сессий	482
Назначение IP-адресов сервисам	482
Анонсирование IP-адресов	482
Распределение трафика	482
Преимущества использования BGP	482
Недостатки использования BGP	482
Пример использования MetalLB в режиме BGP LoadBalancer	482
Дополнительные ресурсы	483
Балансировка в кластерах на облачных платформах	483
Пример создания Ingress-контроллера для провайдера OpenStack	483
Пример создания сервиса с типом ClusterIP	484
Пример создания внутреннего балансировщика для VK Cloud	484
ALB средствами Ingress NGINX Controller	484
Варианты терминации трафика	485
Терминация HTTPS	485
Мониторинг и статистика	485
Статистика	485
Основные принципы сбора статистики	486
Состав и представление метрик	486
Примеры настройки балансировки	486
Пример для AWS (Network Load Balancer)	486
Пример для GCP, Yandex Cloud и Azure	487
Пример для OpenStack	487
Пример создания внутреннего балансировщика для VK Cloud	487
Пример для bare metal	488
Пример для bare metal при использовании внешнего балансировщика	488
Пример для bare metal (MetalLB в режиме BGP LoadBalancer)	488
Пример для bare metal (балансировщик MetalLB в режиме L2 LoadBalancer)	489
Пример разделения доступа между публичной и административной зонами	491
Особенности использования одного Ingress-контроллера	491
Использование отдельных Ingress-контроллеров	492
Поддержка и безопасность модуля	494
Дополнительные ресурсы	494
ALB средствами Kubernetes Gateway API	494
Обзор и схема	494
Роли объектов	495

Зачем нужен ListenerSet	495
ClusterALBInstance и ALBInstance	495
Валидация конфигурации	496
Инлеты	496
Как настроить ALB	496
Действия перед включением и настройкой ALB в кластере	496
Включение модуля и создание Gateway	497
Создание ListenerSet	497
Создание маршрутов	498
Публикация служебных доменов	498
Выбор шлюза DKP по умолчанию при использовании нескольких ClusterALBInstance	499
Смена шлюза DKP по умолчанию	500
Примеры для разных окружений	500
Смена инлета с сохранением текущего Gateway	502
Открытие дополнительного TCP/UDP-порта на общекластерном Gateway	502
Разделение публичной и административной зон	503
Публикация в неймспейсе (ALBInstance)	504
Выпуск TLS-сертификатов с cert-manager	504
Совместное использование с другими модулями и сторонними решениями	504
Совместно с ingress-nginx	504
Совместно со сторонними реализациями Gateway API	505
Включение HTTP/3	505
Использование GeoIP и GeoLite2	506
Использование заголовков GeoIP	507
Настройка трассировки OpenTelemetry	508
Настройка TLS для OpenTelemetry	508
Диагностика и проверка	509
Дополнительные ресурсы	510
Миграция с ingress-nginx на alb	510
Причины перехода на Gateway API	510
Сравнение моделей	510
Ingress API и ingress-nginx	510
Gateway API и alb	511
Ключевые архитектурные различия	511
Шаг 1. Подготовка инфраструктуры модуля alb	513
Выбор ALBInstance или ClusterALBInstance	513
Настройка инлета	513
TLS и сертификаты	514
Шаг 2. Миграция интерфейсов DKP	515
Шаг 3. Миграция публикации приложений	515
Конвертация Ingress в Gateway API	515
Использование встроенного инструмента ingress2gateway	515
Расширение Gateway API с помощью аннотаций	516
Шаг 4. Переключение трафика на модуль alb	517
Тестирование модуля alb через Ingress-контроллер	518
Обеспечение проверки HTTP-01 во время миграции	519
Выбор способа переключения	520
Автоматически создаваемый балансировщик нагрузки	520
Балансировщик нагрузки под ручным управлением	520
Прямой доступ через HostPort или HostWithFailover	521
Проверка переключения	521
Откат	521
Шаг 5. Очистка	522
Дополнительные ресурсы	522
ALB средствами Istio	522
Ingress для публикации приложений	522
Istio Ingress Gateway	522
Ingress NGINX	523
Дополнительные ресурсы	523
Обеспечение высокой доступности и отказоустойчивости (keepalived)	523
Примеры использования модуля	524
Несколько публичных IP-адресов	524

Ручное переключение keepalived	525
Дополнительные ресурсы	525
Настройка Egress Gateway	525
Режимы использования Egress Gateway	525
Базовый режим	525
Режим с Virtual IP	525
Принцип настройки	526
Обслуживание узла	526
Сравнение с CiliumEgressGatewayPolicy	526
Примеры настройки	526
EgressGateway в режиме PrimaryIPFromEgressGatewayNodeInterface (базовый режим)	526
EgressGateway в режиме VirtualIPAddress (Режим с Virtual IP)	527
EgressGatewayPolicy	527
Дополнительные ресурсы	528
Межкластерное взаимодействие (альянс кластеров)	528
Особенности использования service mesh DKP	528
Отличия федерации и мультикластера	528
Модели объединения нескольких кластеров (на примере Istio)	528
Задачи, решаемые с помощью мультикластера и федерации	528
Дополнительные ресурсы	529
Мультикластер	529
Мультикластер средствами Istio	529
Требования к кластерам	529
Общие принципы	529
Включение мультикластера	529
Управление мультикластером	530
Пример устройства мультикластера из двух кластеров	530
Дополнительные ресурсы	530
Федерация	530
Федерация средствами Istio (Service Mesh)	530
Требования к кластерам	530
Общие принципы федерации	531
Включение федерации: создаваемые сервисы	531
Настройка федерации	531
Пример устройства федерации из двух кластеров	531
Дополнительные ресурсы	532
Настройка сетевых политик	532
Настройка сетевых политик стандартными средствами Kubernetes	532
Пример настроек сетевой политики	532
Настройка сетевых политик на уровне всего кластера с помощью CiliumClusterwideNetworkPolicy	533
Дополнительные ресурсы	534
Программно-определяемые сети (SDN)	534
Дополнительные ресурсы	534
Настройка SDN в кластере	534
Подготовка инфраструктуры к включению модуля sdn	534
Включение модуля sdn	535
Действия после включения модуля sdn	535
Разметка интерфейсов для организации дополнительных программно-определяемых сетей	535
Объединение нескольких физических интерфейсов в интерфейс агрегации каналов (bond-интерфейс)	535
Настройка и подключение дополнительных виртуальных сетей для использования в прикладных подах	537
Создание общедоступной (кластерной) сети	537
Создание общедоступной сети, основанной на тегированном трафике	538
Создание сети, основанной на прямом доступе в интерфейс	538
Создание сети проекта (пользовательской сети)	539
Проверка присоединения дополнительной сети к интерфейсам на узлах	539
IPAM для дополнительных сетей	540
Принципы и особенности работы IPAM в DKP	540
Защита от конфликтов при переиспользовании IP-адресов	541
Ресурсы и параметры для настройки IPAM	541

Пример выделения пула IP-адресов для кластерной сети	541
Настройка и подключение underlay-сетей для проброса аппаратных устройств	542
Основные возможности	542
Режимы работы	542
Автоматическая группировка интерфейсов	542
Порядок настройки и подключения физических интерфейсов в прикладные поды	543
Предварительные требования для DPDK-приложений	543
Настройка hugepages	543
Настройка Topology Manager	543
Предварительные действия перед созданием UnderlayNetwork	544
Проверка и настройка сетевых интерфейсов (песчниц NodeNetworkInterface)	544
Создание Underlay-сети в режиме Dedicated	544
Создание Underlay-сети в режиме Shared	545
Подготовка неймспейса для использования UnderlayNetwork	547
Настройка и подключение системных сетей (сервисных сетей)	547
Предварительные требования для создания и использования системных сетей	547
Создание системной сети	547
Тип VLAN	548
Тип Access	548
Тип SRIOVirtualFunction	548
Создание пула IP-адресов для настройки IPAM системной сети	549
Проверка статуса системной сети	549
Дополнительные ресурсы	549
Управление статичными маршрутами и правилами ip rule на узлах кластера	549
Примеры настроек	550
Создание маршрута в основной таблице «main»	550
Создание маршрута в дополнительной таблице	550
Создание ip rule	550
Дополнительные ресурсы	551
Управление DNS в кластере Kubernetes	551
Пример конфигурации	551
Дополнительные ресурсы	552
Кеширование DNS-запросов на узлах кластера	552
Пример настройки кастомного DNS в поде	552
Дополнительные ресурсы	552
Режимы работы bpfLB для обработки внешнего трафика	553
Дополнительные ресурсы	554
Синхронизация времени на узлах	554
Включение встроенной синхронизации времени	554
Использование собственных NTP-демонов	555
Дополнительные ресурсы	555
Доступ через OpenVPN	555
Настройка	556
Доступ к OpenVPN снаружи кластера	556
Пример для bare-metal кластеров	556
Пример для AWS и Google Cloud	556
Пример для публичного IP-адреса на внешнем балансировщике	556
Аутентификация	557
Доступные ресурсы кластера после подключения к VPN	557
Аудит пользовательских соединений	557
Почему не работает автоматическая настройка DNS-сервера при подключении на macOS и Linux с помощью клиента OpenVPN	557
Как отозвать, ротировать или удалить сертификат клиента	558
Как ротировать сертификат сервера	558
Как ротировать корневой сертификат (CA)	558
Дополнительные ресурсы	559
Хранилища	559
Поддерживаемые типы хранилищ	559
Программно-определяемые хранилища	559
Внешние хранилища	559
Ключевые возможности	559
Настройка хранилищ	559

Создание StorageClass	560
Назначение StorageClass по умолчанию	560
Дополнительные ресурсы	560
Настройка создания снимков томов	560
Поддерживаемые CSI-драйверы	560
Создание снимков	560
Восстановление из снимка	561
Дополнительные ресурсы	561
Локальное хранилище Local Path Provisioner	561
Принцип работы	561
Ограничения	562
Примеры ресурсов LocalPathProvisioner	562
ReclaimPolicy: Retain (по умолчанию)	562
ReclaimPolicy: Delete	562
Настройка Prometheus с использованием локального хранилища	562
Дополнительные ресурсы	562
Настройка узлов для программно-определяемых хранилищ	563
Дополнительные ресурсы	563
Примеры использования	563
Работа с ресурсами BlockDevice	563
Создание ресурса BlockDevice	563
Критерии отбора устройства контроллером	563
Обновление ресурса BlockDevice	564
Удаление ресурса BlockDevice	564
Работа с ресурсами LVMVolumeGroup	564
Создание ресурса LVMVolumeGroup	564
Обновление ресурса LVMVolumeGroup	566
Удаление ресурса LVMVolumeGroup	566
Вывод ресурса BlockDevice из LVMVolumeGroup	567
Защита от утечек данных между томами	567
Thick volume	567
Использование режима Discard	567
Thin volume	568
Дополнительные ресурсы	568
Сценарии конфигурации	569
Клонирование виртуальных машин	569
Способы и сценарии конфигурации дисковой подсистемы узлов	569
Хранилище с одинаковыми дисками	570
Полное зеркало	571
Пример настройки модулей SDS (одинаковые диски, «Полное зеркало»)	571
Настройка модуля sds-local-volume (одинаковые диски, «Полное зеркало»)	571
Настройка модуля sds-replicated-volume (одинаковые диски, «Полное зеркало»)	572
Частичное зеркало	573
Пример настройки модулей SDS (одинаковые диски, «Частичное зеркало»)	574
Настройка модуля sds-local-volume (одинаковые диски, «Частичное зеркало»)	574
Настройка модуля sds-replicated-volume (одинаковые диски, «Частичное зеркало»)	575
Комбинированное хранилище	576
Настройка дополнительных дисков (Полное зеркало)	577
Пример настройки модулей SDS (комбинированное хранилище, «Полное зеркало»)	577
Настройка модуля sds-local-volume (комбинированное хранилище, «Полное зеркало»)	578
Настройка модуля sds-replicated-volume (комбинированное хранилище, «Полное зеркало»)	578
Настройка дополнительных дисков (Частичное зеркало)	579
Пример настройки модулей SDS (комбинированное хранилище, «Частичное зеркало»)	580
Настройка модуля sds-local-volume (комбинированное хранилище, «Частичное зеркало»)	580
Настройка модуля sds-replicated-volume (комбинированное хранилище, «Частичное зеркало»)	581
Дополнительные ресурсы	583
FAQ	584
Причины, по которым в кластере не создаются ресурсы BlockDevice	584

Причины, по которым в кластере не создаются ресурсы LVMVolumeGroup	584
Причины, по которым после удаления LVMVolumeGroup ресурс и Volume Group остаются	584
Причины неуспешного создания Volume Group с помощью ресурса LVMVolumeGroup	584
Поведение ресурса LVMVolumeGroup при отключении одного из устройств в Volume Group	584
Передача управления существующей на узле Volume Group контроллеру	584
Отключение отслеживания LVM Volume Group контроллером	585
Причины автоматической установки тега storage.deckhouse.io/enabled=true для Volume Group	585
Использование ресурса LVMVolumeGroupSet для создания LVMVolumeGroup	585
Лейблы, добавляемые контроллером к ресурсам BlockDevice	585
Дополнительные ресурсы	586
Настройка локального хранилища на основе LVM	586
Настройка локального хранилища	586
Быстрый старт	586
Включение модулей	586
Подготовка узлов к созданию хранилищ	587
Настройка хранилища на узлах	587
Выбор метода очистки тома после удаления PV	590
Thick-тома	591
Thin-тома	591
Системные требования и рекомендации	591
Дополнительные ресурсы	591
Управление локальным хранилищем на основе LVM	591
Выбор конкретных узлов для использования модуля	591
Проверка создания PVC на выбранном узле	592
Вывод узла из-под управления модуля	592
Проверка зависимых ресурсов LVMVolumeGroup на узле	593
Оставшийся под sds-local-volume-csi-node после удаления лейблов	594
Отсутствие служебных подов на нужном узле	594
Перемещение данных между PVC	595
Создание снимков томов	596
Дополнительные ресурсы	597
Настройка реплицируемого хранилища на основе DRBD	597
Включение модуля	597
Обнаружение компонентов LVM	597
Подключение DRBD	598
Преднастройка узлов	598
Создание групп томов LVM	598
Создание реплицированных thick pool	600
Создание реплицированных thin pool	600
Создание объектов StorageClass	601
Дополнительные ресурсы	602
Управление реплицируемым хранилищем на основе DRBD	602
Системные требования	603
Рекомендации	603
Получение информации об использовании пространства	603
Назначение StorageClass по умолчанию	603
Добавление существующей LVMVolumeGroup	603
Изменение ограничений DRBD-томов и портов кластера	603
Корректная перезагрузка узла с DRBD-ресурсами	604
Перемещение ресурсов для освобождения места в Storage Pool	604
Автоматизированное управление репликами и мониторинг состояния LINSTOR	605
Вытеснение DRBD-ресурсов с узла	605
Подготовка и запуск скрипта	605
Пример удаления узла из LINSTOR и Kubernetes	606
Пример удаления ресурсов с узла	606
Параметры скрипта evict.sh	606
Диагностика проблем	607
Ошибка запуска linstor-node при загрузке DRBD-модуля	607
Ошибка FailedMount при запуске пода	607

При зависании пода на стадии ContainerCreating	607
Ошибки Input/output error	607
После удаления ресурса ReplicatedStoragePool остаётся соответствующий ему Storage Pool	608
Ограничения на изменение spec ReplicatedStorageClass	608
Удаление дочернего StorageClass при удалении ReplicatedStorageClass	608
Ошибки при создании Storage Pool или StorageClass	608
Ошибка “You’re not allowed to change state of linstor cluster manually”	608
Восстановление БД из резервной копии	608
Процесс восстановления резервной копии	608
Отсутствие служебных подов sds-replicated-volume на выбранном узле	610
Дополнительная поддержка	611
Миграция с модуля linstor на sds-replicated-volume	611
Порядок действий для миграции	611
Миграция на ReplicatedStorageClass	612
Миграция на ReplicatedStoragePool	613
Миграция с модуля sds-drbd на sds-replicated-volume	613
Порядок действий для миграции	613
Причины отказа от RAID с sds-replicated-volume	615
Рекомендации по использованию локальных дисков	615
Ручной запуск процесса перевыпуска сертификатов	615
Дополнительные ресурсы	616
Распределённое хранилище Ceph	616
Миграция с модуля ceph-csi	616
Подключение к Ceph-кластеру	616
Дополнительная информация	618
Получение списка томов RBD, разделённого по узлам	618
Поддерживаемые версии Ceph-кластеров	618
Поддерживаемые режимы работы томов	618
Разрешения (caps) для пользователей в Ceph	618
RBD	618
CephFS	619
CephFS + RBD	619
Дополнительные ресурсы	619
Хранилище данных HPE	619
Системные требования	620
Настройка	620
Включение модуля	620
Создание StorageClass	620
Проверка работоспособности модуля	621
Дополнительные ресурсы	621
Хранилище данных Huawei	621
Системные требования	621
Настройка	621
Включение модуля	621
Создание StorageClass	622
Проверка работоспособности модуля	623
Дополнительные ресурсы	623
Хранилище данных NetApp	623
Системные требования	623
Настройка интеграции кластера с системой хранения NetApp	623
Дополнительные ресурсы	624
Сетевое файловое хранилище NFS	624
Системные требования	624
Минимальные требования	624
Рекомендации по настройке	625
Ограничения	625
Общие ограничения	625
Ограничения снимков томов	625
Ограничения режима RPC-with-TLS	625
Функциональные ограничения	625
Требования к системе	625

Настройка	626
Включение модуля	626
Создание StorageClass	626
Настройка очистки томов	627
Параметр RandomFillSinglePass	627
Параметр RandomFillThreePass	627
Параметр Discard	627
Изменение параметров NFS-сервера для созданных PersistentVolume	627
Создание снимков томов	627
Проблемы с удалением PV с поддержкой RPC-with-TLS	628
Добавление нескольких CA в параметр tlsParameters.ca	628
Дополнительные ресурсы	628
Объектное хранилище на основе S3	628
Системные требования	629
Настройка	629
Включение модуля	629
Создание StorageClass	630
Проверка работоспособности модуля	630
Ограничения	630
Известные ошибки и проблемы	631
Изменение параметров подключения для существующих PV	631
Особенность отображения размера монтированной директории	631
Обработка превышения квоты bucket или пользователя	631
Получение информации об используемом пространстве	631
Использование нескольких S3-хранилищ в одном модуле	631
Использование одного bucket для нескольких подов	632
Диагностика неисправностей	632
Проблемы при создании PVC	632
Проблемы с запуском контейнеров	632
Дополнительные ресурсы	632
Хранилище данных на основе протокола SCSI	632
Поддерживаемые возможности	632
Ограничения	633
Системные требования	633
Быстрый старт	633
Включение модуля csi-scsi-generic	633
Создание SCSTarget	633
iSCSI	633
Fibre Channel	634
Создание StorageClass	635
Проверка работоспособности модуля csi-scsi-generic	636
Дополнительные ресурсы	636
Унифицированное хранилище TATLIN.UNIFIED (Yadro)	636
Включение модуля	636
Подключение к системе хранения данных TATLIN.UNIFIED	636
Создание StorageClass	637
Проверка работоспособности модуля	637
Дополнительные ресурсы	637
Безопасность	637
Дополнительные ресурсы	638
События безопасности	638
Зависимости и требования	638
Источники данных для событий безопасности	638
Включение событий безопасности	639
Настройка источников событий	639
Настройка отправки событий	640
Настройка приёмника	640
Настройка правил отправки событий	641
Стандартные настройки модуля	641
Поддерживаемые события безопасности	642
Дополнительные ресурсы	642
Аудит событий API Kubernetes	642

Встроенные политики аудита	642
Настройка собственной политики аудита	643
Как формировать содержимое файла audit-policy.yaml	644
Примеры	645
Логирование всех запросов от всех аутентифицированных пользователей	645
Логирование всех запросов не от сервисных аккаунтов	645
Логирование всех операций с типовыми ресурсами	646
Работа с лог-файлом аудита	646
Перенаправление лог-файла аудита в stdout	647
Дополнительные ресурсы	647
Рантайм-аудит	647
Источники данных для рантайм-аудита	647
Минимальные требования	647
Включение рантайм-аудита	648
Настройка вебхука API Kubernetes вручную	648
Работа с правилами аудита	649
Добавление пользовательского правила	649
Применение стороннего правила	650
Сбор логов и оповещения	651
Настройка сбора логов и событий	651
Настройка оповещений о критических событиях	651
Просмотр метрик	652
Отладка и эмуляция событий	652
Включение логов для отладки	652
Настройка эмуляции событий	652
Falco	653
Falcosekick	653
Дополнительные ресурсы	653
Политики безопасности	653
Применение Pod Security Standards	654
Политика по умолчанию	654
Назначение политики	654
Режимы применения политики	654
Расширение политики	654
Операционные политики	656
Политики безопасности	656
Частичное применение политик	659
Кастомные ресурсы Gatekeeper	661
Примеры пользовательских политик Gatekeeper	661
Запрет на удаление узла без указанного лейбла	661
Запрет на выполнение операций exec и attach в определённые поды	662
Встроенная политика для подов с лейблом heritage: deckhouse	662
Пример пользовательской политики	663
Проверка подписи образов	664
Использование альтернатив для управления политиками безопасности	666
Дополнительные ресурсы	666
Сканирование контейнерных образов на уязвимости	666
Поиск уязвимостей	666
Сканирование в пространствах имён	666
Условия запуска и алгоритм сканирования	667
Принудительный повтор сканирования	667
Управление сертификатами	667
Мониторинг	668
Роли доступа	668
Работа с издателями сертификатов	668
Добавление ClusterIssuer с валидацией DNS-01 через вебхук	668
Добавление ClusterIssuer, использующего собственный удостоверяющий центр (CA)	669
Добавление Issuer и ClusterIssuer, использующих HashiCorp Vault для заказа сертификатов	670
Дополнительные ресурсы	671
Интеграция с ПО и решениями для безопасности	671
Отправка логов в KUMA	671

Примеры конфигурации ClusterLogDestination и ClusterLoggingConfig	671
Особенности настройки KCS	675
Исключения при антивирусном сканировании узлов	675
Рекомендации по настройке KESL	675
Дополнительные ресурсы	676
Автоматическое распространение секретов по пространствам имён	676
Как это работает	676
Несовместимость с режимом мультитенантности	676
Настройка автоматического распространения секретов	676
Распространение секретов по выбранным пространствам имён	677
Мониторинг в Deckhouse Kubernetes Platform	677
Мониторинг аппаратных ресурсов	677
Мониторинг Kubernetes	677
Мониторинг Ingress	677
Отключение сбора детализированной статистики Ingress-ресурсов	678
Мониторинг control plane	678
Мониторинг кластера	678
Режим расширенного мониторинга	678
Алерты	679
Отправка алертов во внешние системы	679
Мониторинг SLA кластера	679
Дополнительные ресурсы	679
Настройка мониторинга сетевого взаимодействия и узлов кластера	679
Мониторинг сетевого взаимодействия	679
Добавление дополнительных IP-адресов для мониторинга	680
Мониторинг узлов кластера	680
Дополнительные ресурсы	680
Запись данных Prometheus в долговременное хранилище	680
Пример минимального PrometheusRemoteWrite	681
Пример расширенного PrometheusRemoteWrite	681
Дополнительные ресурсы	681
Настройка внешнего доступа	681
Подключение Prometheus к сторонней Grafana	681
Подключение стороннего приложения к Prometheus	683
Сбор метрик через шлюз (Pushgateway)	684
Удаление метрик из шлюза (Pushgateway)	685
Дополнительные ресурсы	686
Настройка системы сбора и хранения метрик	686
Что делает Prometheus?	686
Как работает Prometheus?	686
Дополнительные ресурсы	686
Мониторинг SLA кластера	687
Настройка модуля	687
Интерфейс	687
Экспорт метрик статуса	688
Аутентификация	689
Особенности работы подов upmeter	689
Дополнительные ресурсы	690
Настройка интеграций	690
Перенаправление алертов в Zabbix	690
Установка	690
Устранение неполадок	693
Отправка алертов в Telegram	693
Пример отправки алертов в Slack с фильтром	694
Пример отправки алертов в Opsgenie	695
Пример отправки алертов по электронной почте	696
Дополнительные ресурсы	697
Логирование	697
Сбор и доставка логов	698
Настройка сбора и доставки логов	698
Интеграция с внешними системами	699
Elasticsearch	699

Использование шаблона индексов	699
Поддержка Elasticsearch < 6.X	700
Splunk	700
Logstash	701
Graylog	701
Формат сообщений	701
Syslog	702
CEF	702
Преобразование логов	702
Преобразование записи в структурированный объект	703
Замена лейблов	703
Удаление лейблов	704
Фильтрация логов	705
Сборка логов из определенного контейнера	705
Сборка логов без заданной строки	705
Аудит событий kubelet	706
Системные логи DKP	706
Буферизация логов	707
Пример включения буферизации в оперативной памяти	707
Пример включения буферизации на диске	707
Пример определения поведения при переполнении буфера	708
Отладка и расширенные возможности	708
Включение debug-логов агента log-shipper	708
Дополнительная информация о каналах передачи логов	708
Обзор топологии	709
Мониторинг нагрузки на каналы	709
Получение необработанных и промежуточных логов	709
Отладка VRL-правил	709
Добавление поддержки нового source или sink	710
Особые случаи	710
Сбор логов с продуктовых пространств имён через опцию labelSelector	710
Лейбл для исключения подов и пространств имён	710
Дополнительные ресурсы	711
Кратковременное хранение логов	711
Интеграция с Grafana Cloud	711
Переход с Grafana Promtail	712
Дополнительные ресурсы	712
Мультитенантность	712
Ограничения	712
Дополнительные ресурсы	713
Управление проектами	713
Создание проекта	713
Автоматическое создание проекта для неймспейса	714
Создание собственного шаблона для проекта	715
Использование лейблов для управления ресурсами	715
Пропуск создания лейбла heritage: multitenancy-manager	715
Исключение ресурсов из управления multitenancy-manager	716
Реализация валидации изменений объектов с помощью пользовательского лейбла	716
Принцип работы валидации в multitenancy-manager	716
Создание собственной валидации	716
Выдача кластерных ресурсов проектам	718
Дополнительные ресурсы	719
Обзор	719
Как это работает	720
Managed PostgreSQL	720
Включение сервиса	720
Зависимости для отдельных функций	720
Пример создания PostgresClass	720
Изменение и удаление PostgresClass	723
Ограничение ресурсов CPU и памяти	724
Управление отказоустойчивостью через зоны доступности	725
Автоматическая проверка настроек PostgreSQL	726

Задание настроек по умолчанию и их переопределение	726
Автоматический расчёт workMem	727
Привязка к выделенным узлам	727
Дополнительные ресурсы	728
Резервное копирование и восстановление	728
Ручное восстановление кластера	728
Восстановление кластера с одним control-plane узлом	728
Восстановление мультимастерного кластера	729
Восстановление отдельных объектов	730
Восстановление объектов Kubernetes из резервной копии etcd	730
Пример шагов по восстановлению объектов из резервной копии etcd	730
Восстановление объектов кластера из выгруженных YAML-файлов	733
Восстановление объектов при смене IP-адреса master-узла	733
Автоматизированная выгрузка объектов при смене IP-адреса	734
Ручное восстановление объектов при смене IP-адреса	735
Создание резервных копий с помощью Deckhouse CLI	736
Резервное копирование etcd	737
Автоматическое резервное копирование etcd	737
Резервное копирование конфигурации кластера	738
Выгрузка логов из Loki	739
Дополнительные ресурсы	740
Автоматическое назначение лейблов и аннотаций пространствам имён	740
Как это работает	741
Настройка автоматического назначения лейблов и аннотаций	741
Пример конфигурации	741
Дополнительные ресурсы	741
Доставка приложений с помощью Argo CD	741
Дополнительные ресурсы	742
Запуск Argo CD	742
Подготовка к запуску Argo CD	742
Развёртывание экземпляра Argo CD	744
Развёртывание нескольких экземпляров Argo CD	746
Расширенные настройки	746
Включение режима высокой доступности	746
Предоставление доступа к кластерным ресурсам	746
Использование собственного домена кластера	749
Дополнительные ресурсы	749
Настройка аутентификации и авторизации	749
Локальная аутентификация	749
Создание дополнительных локальных пользователей	750
Создание пользовательского токена	750
Отключение локальной аутентификации	751
Аутентификация с помощью SSO	751
Использование самоподписного сертификата	752
Создание пользовательского токена	753
Дополнительные ресурсы	753
Настройка ролевой модели доступа	755
Встроенные роли	755
Политика по умолчанию для аутентифицированных пользователей	755
Анонимный доступ	756
Структура ролевой модели доступа	756
Привязка к роли	756
Политика доступа	756
Ресурсы и поддерживаемые действия	757
Политики, привязанные к приложениям	757
Ресурс applications	757
Детализированные разрешения для update и delete	758
Действие action	758
Действие override	758
Другие ресурсы	759
Ресурс applicationsets	759
Ресурс logs	759

Ресурс exes	759
Ресурс extensions	759
Эффект deny	759
Порядок проверки политик и режимы сопоставления	759
Сопоставление в режиме glob	759
Использование пользователей и групп SSO	760
Локальные пользователи	760
Тестирование политик RBAC	761
Дополнительные ресурсы	761
Marketplace	761
Задачи администратора	761
Ключевые ресурсы	761
Дополнительные ресурсы	762
Репозитории пакетов	762
Управление аутентификацией и интервалом сканирования	762
Аутентификация	762
Интервал сканирования	762
Проверка состояния репозитория	763
Просмотр обнаруженных версий пакетов	763
Дополнительные ресурсы	764
Сканирование	764
Просмотр операций сканирования	764
Детали операции сканирования	764
Запуск сканирования вручную	765
Флаг fullScan	765
Дополнительные ресурсы	766
Обновление платформы	766
Настройка обновлений	766
Проверка текущего канала обновлений	766
Смена канала обновлений	766
Режимы обновления	767
Только патч-версии в рамках текущей минорной (AutoPatch)	767
Автоматическое обновление всех версий (Auto)	767
Ручной режим (Manual)	767
Проверка текущего режима обновления	768
Как применяется автоматическое обновление	768
Закрепление релиза	768
Ручное подтверждение обновлений	769
Окна обновлений	770
Установка обновлений при настроенных окнах обновлений	770
Настройка окон обновлений	770
Примеры конфигурации	770
Дополнительные ресурсы	770
Настройка уведомлений о новых релизах	771
Алерты в системе мониторинга	771
Настройка уведомлений	771
Поддерживаемые параметры	773
Формат уведомлений	773
Дополнительные ресурсы	774
Обновление Kubernetes и управление версиями	774
Обновление и управление версиями	774
Изменение версии Kubernetes	774
Мониторинг процесса обновления Kubernetes	775
Примеры содержимого ConfigMap	775
Кластер в актуальном состоянии (3 master-узла, 3 worker-узла)	775
Начало обновления (например, понижение версии Kubernetes)	776
Обновление control plane в процессе	777
Промежуточный шаг многошагового обновления (например, 1.33 → 1.35)	778
Кластер в актуальном состоянии (2 master-узла, 1 arbitr-узел и 3 worker-узла)	780
Сбой одного или нескольких компонентов control plane	780
Дополнительные ресурсы	781
Удаление	781

Удаление кластера, развёрнутого в облачном провайдере	781
Удаление гибридного кластера	782
Удаление статического кластера	783
Дополнительные ресурсы	784
Архитектура	784
Архитектура DKP	784
Модули	784
Дополнительные ресурсы	785
Архитектура DKP (модель C4)	785
Условные обозначения	785
Диаграмма контекста	786
Диаграмма подсистем	787
Диаграмма модулей	788
Дополнительные ресурсы	789
Разработка модуля Deckhouse Kubernetes Platform	789
Требования	790
Прежде чем начать	790
Вопросы по документации	790
Структура модуля	790
charts	792
crds	792
docs	793
hooks	794
images	795
openapi	795
conversions	795
Тесты конверсий	796
config-values.yaml	796
Валидации x-deckhouse-validations (CEL)	796
Примеры правил	797
Валидация скалярных значений и массивов	797
Валидация additionalProperties (map)	798
values.yaml	798
templates	798
.helmignore	799
Chart.yaml	799
module.yaml	799
Настройка доступности модуля в редакциях DKP	801
Логика определения доступности модуля	802
Примеры конфигурации	802
Дополнительные ресурсы	803
Сборка и публикация модуля	803
Состав артефактов модуля	803
Сборка артефактов модуля и публикация в хранилище образов	803
Дополнительные ресурсы	804
Запуск и проверка модуля в кластере	804
Запуск модуля в кластере DKP	805
Источник модулей	805
Переключение модуля на другой источник модулей	809
Политика обновления модуля	809
Включение модуля в кластере	810
Если что-то пошло не так	810
Подключение Deckhouse Module Tools для проверки модуля	811
Для GitHub-проектов	811
Для GitLab-проектов	812
Шаги для подключения	812
Дополнительные ресурсы	812
Зависимости модуля Deckhouse Kubernetes Platform	812
Зависимость от версии Deckhouse Kubernetes Platform	812
Зависимость от версии Kubernetes	813
Зависимость от версии других модулей	814
Обязательные зависимости	814

Условные зависимости	815
Ограничения по включению и выключению prometheus при наличии условной зависимости от test	815
Ограничения по обновлению prometheus при наличии условной зависимости от test	815
Ограничения по включению test, версия которого указана в зависимостях prometheus	816
Ограничения по обновлению test, версия которого указана в зависимостях prometheus	816
Версионирование модуля	816
Каналы обновлений	816
Порядок публикации новой версии	816
Жизненный цикл модуля	817
Определение стабильности модуля	817
Версионирование API	817
Выпуск новой версии API	818
Автоматическая конвертация API	818
Рекомендации по выпуску новых версий CRD	818
Дополнительные ресурсы	818
Пример адаптации существующего чарта	818
Подготовка исходного кода модуля и сборки	818
Публикация модуля на канале обновлений	820
Подключение модуля в кластере	820
Миграция ModuleUpdatePolicy на версию v1alpha2	822
Дополнительные ресурсы	822
Разработка и отладка модуля	822
Доступность модуля и включение по умолчанию	824
Как работает ModulePullOverride	824
Пример	825
Логика автообновления модулей	827
Пропуск промежуточных релизов (from-to)	827
Примеры	829
Артефакты модуля в хранилище образов	830
Вывод списка модулей в источнике модулей	831
Вывод списка образов модуля	831
Вывод файлов в образе модуля	831
Вывод списка образов контейнеров приложений модуля	831
Конфигурация дополнительных образов	832
Просмотр списка релизов	832
Вывод версии, используемой на канале обновлений alpha	832
Дополнительные ресурсы	832
Катастрофоустойчивость	832
Геораспределённость	833
Использование нескольких зон доступности (Multi-AZ)	833
Характеристики зон доступности	833
Распределение узлов по зонам	833
Использование нескольких регионов (Multi-Region)	833
Характеристики регионов	834
Распределение узлов по разным регионам	834
Балансировка входящего трафика	834
Организация хранилища	834
Георезервирование	834
Балансировка входящего трафика в мультикластере	834
Балансировка по схеме active-active	835
Балансировка по схеме active-standby	835
Организация хранилища в мультикластере	836
Обновление	836
Релизы	836
Каналы обновлений	836
Процесс переключения на более стабильный канал	837
Процесс переключения на менее стабильный канал	837
Обновление control plane	837
Обновление патч-версий	837

Автоматическое обновление минорных версий	837
Ручное обновление минорных версий	837
Процесс обновления компонентов control plane	838
Получение истории изменений (changelog)	838
Содержимое changelog	838
Минорные версии и нулевая патч-версия	838
Проверка зависимостей при обновлении	838
Дополнительные ресурсы	839
Подсистема Deckhouse	839
Модули веб-интерфейса	839
Модуль console	839
Архитектура модуля	839
Компоненты модуля	840
Взаимодействия модуля	840
Модуль documentation	841
Архитектура модуля	841
Компоненты модуля	842
Взаимодействия модуля	842
Модуль deckhouse-tools	842
Архитектура модуля	842
Компоненты модуля	843
Взаимодействия модуля	843
Дополнительные ресурсы	843
Модуль deckhouse	843
Архитектура модуля	844
Компоненты модуля	844
Взаимодействия модуля	845
Дополнительные ресурсы	845
Механизм очередей	846
Очереди addon-operator	846
Очереди Marketplace	847
Дополнительные ресурсы	848
Модуль registry	848
Архитектура модуля	849
Компоненты модуля	852
Взаимодействия модуля	853
Дополнительные ресурсы	853
Архитектура режима Direct	853
Дополнительные ресурсы	854
Marketplace	854
Разделы	854
Концепции	854
Типы Package	854
Модель ресурсов	855
Содержимое ApplicationPackageVersion	855
Жизненный цикл от сканирования до деплоя	855
Ограничения Application	856
Функциональные ограничения	856
Ограничения на имена	856
Дополнительные ресурсы	856
Разработка приложений	856
Предварительные условия	856
Бутстрап нового Application	856
Структура проекта	857
package.yaml	857
Локальная сборка	858
Линтинг	858
Настройка CI/CD	858
Переменные окружения	858
Выпуск релиза	858
Организация OCI-артефактов в реестре	858
Содержимое bundle	859

Содержимое образа метаданных	859
Аннотации Nelm	859
Стадии деплоя	859
1. Render	859
2. Plan	859
3. Apply	859
Типичные задачи	860
1. Job, которая должна запускаться перед основным приложением	860
2. Сохранение ресурса при удалении или uninstall	860
3. Ресурс, общий между релизами	860
4. Зависимость от ресурса, созданного оператором	861
5. Некритичный компонент, не роняющий весь деплой	861
6. Ресурс, рендерящийся только при первой установке	861
7. Медленно стартующий ресурс с большим образом	861
Порядок и зависимости	862
Веса	862
Прямые зависимости	862
Внешние зависимости	862
Зависимости при удалении	863
Аннотации жизненного цикла	863
helm.sh/resource-policy	863
werf.io/ownership	863
werf.io/deploy-on	863
werf.io/delete-policy	863
werf.io/delete-propagation	863
Аннотации трекинга	864
Аннотации логов	864
Шаблонные функции	864
werf_secret_file	864
dump_debug, printf_debug, include_debug, tpl_debug	864
Известные подводные камни	865
Хуки	865
ApplicationHookInput	865
Пример: базовый хук	865
Хук валидации настроек	865
Гарантия изоляции неймспейса	866
Подсистема Delivery	866
Модуль operator-argo	866
Архитектура модуля	867
Компоненты модуля	867
Взаимодействия модуля	871
Дополнительные ресурсы	871
Подсистема Kubernetes & Scheduling	871
Control plane кластера	871
Kubelet	872
Манифесты статических подов	872
Взаимодействия kubelet	872
Дополнительные ресурсы	873
Управление компонентами control plane кластера	874
Модуль control-plane-manager	874
Архитектура модуля	874
Компоненты модуля	875
Взаимодействия модуля	875
Мониторинг control plane кластера	876
Компоненты модуля monitoring-kubernetes-control-plane	876
Взаимодействия компонента control-plane-proxy	876
Сбор метрик с kube-apiserver	876
Дополнительные ресурсы	876
Вертикальное масштабирование	876
Режимы работы VPA	877
Ограничения VPA	877
Архитектура модуля	877

Компоненты модуля	879
Взаимодействия модуля	880
Дополнительные ресурсы	880
Модуль adaptive-resource-management	880
Архитектура модуля	881
Компоненты модуля	881
Взаимодействия модуля	881
Дополнительные ресурсы	881
Модуль descheduler	882
Архитектура модуля	882
Компоненты модуля	882
Взаимодействия модуля	882
Дополнительные ресурсы	882
Модуль pod-reloader	882
Архитектура модуля	882
Компоненты модуля	883
Взаимодействия модуля	883
Дополнительные ресурсы	883
Подсистема Cluster & Infrastructure	884
Управление узлами	884
Типы узлов и механика добавления	884
Автоматическое развертывание, настройка и обновление узлов Kubernetes	885
Развертывание узлов Kubernetes	885
Поддержка актуального состояния узлов	885
Работа с узлами в поддерживаемых облаках	885
Масштабирование узлов в облаке	885
Работа со статическими узлами	886
Группировка узлов и управление группами	886
Что такое ресурс Instance	886
Когда требуется перезагрузка узлов	887
Влияние параметров NodeGroup на обновление и перезапуск узлов	887
Дополнительные ресурсы	887
Bashible	887
Bashible-скрипты и служба bashible	887
Bashible-api-server	888
Модуль node-manager	888
Архитектура модуля	888
Дополнительные ресурсы	888
Управление CloudEphemeral-узлами	889
Архитектура модуля	889
Компоненты модуля	889
Взаимодействия модуля	890
Особенности архитектуры, специфичные для CloudEphemeral-узлов	891
Дополнительные ресурсы	891
Управление CloudPermanent-узлами	891
Архитектура модуля	891
Компоненты модуля	892
Взаимодействия модуля	893
Особенности архитектуры, специфичные для CloudPermanent-узлов	893
Дополнительные ресурсы	893
Управление CloudStatic-узлами	893
Архитектура модуля	893
Компоненты модуля	894
Взаимодействия модуля	895
Особенности архитектуры, специфичные для CloudStatic-узлов	895
Дополнительные ресурсы	895
Управление Static-узлами	895
Архитектура модуля	895
Компоненты модуля	896
Взаимодействия модуля	897
Особенности архитектуры, специфичные для Static-узлов	897
Дополнительные ресурсы	898

Управление гибридными группами узлов и кластерами	898
Дополнительные ресурсы	898
Модуль cloud-provider-aws	898
Архитектура модуля	898
Компоненты модуля	899
Взаимодействия модуля	900
Дополнительные ресурсы	901
Модуль cloud-provider-azure	901
Архитектура модуля	901
Компоненты модуля	901
Взаимодействия модуля	902
Дополнительные ресурсы	903
Модуль cloud-provider-dvp	903
Архитектура модуля	903
Компоненты модуля	904
Взаимодействия модуля	905
Дополнительные ресурсы	906
Модуль cloud-provider-dynamix	906
Архитектура модуля	906
Компоненты модуля	907
Взаимодействия модуля	908
Дополнительные ресурсы	909
Модуль cloud-provider-gcp	909
Архитектура модуля	909
Компоненты модуля	910
Взаимодействия модуля	911
Дополнительные ресурсы	911
Модуль cloud-provider-huaweicloud	911
Архитектура модуля	911
Компоненты модуля	912
Взаимодействия модуля	913
Дополнительные ресурсы	914
Модуль cloud-provider-openstack	914
Архитектура модуля	914
Компоненты модуля	915
Взаимодействия модуля	916
Дополнительные ресурсы	916
Модуль cloud-provider-vcd	916
Архитектура модуля	917
Компоненты модуля	918
Взаимодействия модуля	919
Дополнительные ресурсы	920
Модуль cloud-provider-vmware	920
Архитектура модуля	920
Компоненты модуля	921
Взаимодействия модуля	922
Дополнительные ресурсы	922
Модуль cloud-provider-yandex	922
Архитектура модуля	922
Компоненты модуля	924
Взаимодействия модуля	925
Дополнительные ресурсы	926
Модуль cloud-provider-zvirt	927
Архитектура модуля	927
Компоненты модуля	928
Взаимодействия модуля	929
Дополнительные ресурсы	930
Модуль chrony	930
Архитектура модуля	930
Компоненты модуля	931
Взаимодействия модуля	932
Дополнительные ресурсы	932

Модуль gru	932
Режим DRA	932
Архитектура модуля	932
Компоненты модуля	933
Взаимодействия модуля	934
Режим Device Plugin	934
Архитектура модуля	934
Компоненты модуля	934
Взаимодействия модуля	935
Дополнительные ресурсы	936
Модуль registry-packages-proxy	936
Архитектура модуля	936
Компоненты модуля	937
Взаимодействия модуля	937
Дополнительные ресурсы	937
Модуль terraform-manager	937
Архитектура модуля	937
Компоненты модуля	938
Взаимодействия модуля	939
Дополнительные ресурсы	939
Подсистема IAM	939
Аутентификация	939
Подключение к API Kubernetes с помощью сгенерированного kubeconfig	939
Защита Dex от подбора логина и пароля	939
Аутентификация с помощью DexAuthenticator	939
Расширения Dex в DKP	940
Отказоустойчивый режим	940
Дополнительные ресурсы	940
Мультитенантность	940
Внутренняя логика работы	940
Создание проекта	940
Изоляция проекта	941
Архитектура модуля	941
Компоненты модуля	941
Взаимодействия модуля	942
Дополнительные ресурсы	942
Модуль user-authn	942
Архитектура модуля	942
Компоненты модуля	945
Взаимодействия модуля	945
Дополнительные ресурсы	946
Подсистема Security	946
Контроль целостности	946
Контроль целостности образов на уровне CRI	946
Контроль целостности пользовательской нагрузки при запуске	947
Контроль целостности работающих контейнеров	947
Защита целостности модулей DKP	947
Дополнительные ресурсы	948
Архитектура рантайм-аудита	948
Правила аудита	948
Встроенные правила	948
Пользовательские правила	949
Дополнительные ресурсы	949
Архитектура событий безопасности	949
Архитектура решения	949
Пайплайн обработки	949
Слой сбора (log-shipper)	950
Обработка и извлечение событий	950
Преобразование и обогащение события	951
Фильтрация и доставка событий	951
Модель события безопасности	951
Минимально достаточный цикл работы	952

Поддерживаемые события безопасности	952
Дополнительные ресурсы	952
Модуль admission-policy-engine	952
Архитектура модуля	953
Компоненты модуля	954
Взаимодействия модуля	955
Дополнительные ресурсы	955
Модуль cert-manager	955
Архитектура модуля	955
Компоненты модуля	956
Взаимодействия модуля	957
Дополнительные ресурсы	958
Модуль operator-trivy	958
Архитектура модуля	959
Компоненты модуля	959
Взаимодействия модуля	961
Дополнительные ресурсы	961
Модуль runtime-audit-engine	961
Архитектура модуля	962
Компоненты модуля	962
Взаимодействия модуля	963
Дополнительные ресурсы	963
Модуль secrets-store-integration	963
Архитектура модуля	963
Компоненты модуля	964
Взаимодействия модуля	965
Дополнительные ресурсы	965
Подсистема Network	965
Модуль cni-cilium	966
Архитектура модуля	966
Компоненты модуля	967
Взаимодействия модуля	969
Дополнительные ресурсы	969
Модуль cilium-hubble	969
Архитектура модуля	970
Компоненты модуля	970
Взаимодействия модуля	971
Дополнительные ресурсы	971
Модуль ingress-nginx	971
Архитектура модуля	971
Компоненты модуля	972
Взаимодействия модуля	973
Способы приема трафика из внешней сети	973
Архитектура Ingress-контроллера с инлетом HostWithFailover	974
Компоненты failover Ingress-контроллера	974
Взаимодействия failover Ingress-контроллера	975
Дополнительные ресурсы	975
Модуль metallb	975
Архитектура модуля	976
Компоненты модуля	977
Взаимодействия модуля	977
Дополнительные ресурсы	978
Модули управления DNS	978
Модуль kube-dns	978
Архитектура модуля	978
Компоненты модуля	979
Взаимодействия модуля	979
Модуль node-local-dns	980
При использовании cni-cilium	980
Архитектура модуля	980
Компоненты модуля	981
Взаимодействия модуля	982

При использовании cni-flannel или cni-simple-bridge	982
Архитектура модуля	982
Компоненты модуля	982
Взаимодействия модуля	983
Дополнительные ресурсы	983
Архитектура Istio на уровне кластера	983
Накладные расходы	985
Control plane	985
Data plane	985
Дополнительные ресурсы	985
Архитектура прикладного сервиса с включенным Istio	985
Особенности архитектуры	985
Жизненный цикл пользовательского запроса	986
Приложение с выключенным Istio	986
Приложение с включенным Istio	986
Дополнительные ресурсы	986
Кеширующий DNS-сервер в кластере	986
Принцип работы кеширующего DNS-сервера	987
Особенности конфигурации CoreDNS	987
Дашборд Grafana	987
Дополнительные ресурсы	987
Подсистема Storage	987
Модуль local-path-provisioner	987
Архитектура модуля	988
Компоненты модуля	988
Взаимодействия модуля	989
Дополнительные ресурсы	989
Модуль snapshot-controller	989
Архитектура модуля	989
Компоненты модуля	990
Взаимодействия модуля	991
Дополнительные ресурсы	991
Модуль sds-local-volume	991
Архитектура модуля	991
Компоненты модуля	992
Взаимодействия модуля	993
Дополнительные ресурсы	993
Модуль sds-node-configurator	993
Архитектура модуля	993
Компоненты модуля	996
Взаимодействия модуля	996
Дополнительные ресурсы	996
Модуль csi-ceph	996
Архитектура модуля	996
Компоненты модуля	997
Взаимодействия модуля	998
Дополнительные ресурсы	998
Модуль csi-hpe	998
Архитектура модуля	998
Компоненты модуля	999
Взаимодействия модуля	1000
Дополнительные ресурсы	1000
Модуль csi-huawei	1000
Архитектура модуля	1000
Компоненты модуля	1001
Взаимодействия модуля	1002
Дополнительные ресурсы	1002
Модуль csi-netapp	1002
Архитектура модуля	1002
Компоненты модуля	1003
Взаимодействия модуля	1003
Дополнительные ресурсы	1004

Модуль csi-nfs	1004
Архитектура модуля	1004
Компоненты модуля	1005
Взаимодействия модуля	1006
Дополнительные ресурсы	1006
Модуль csi-s3	1006
Архитектура модуля	1006
Компоненты модуля	1007
Взаимодействия модуля	1007
Дополнительные ресурсы	1008
Модуль csi-scsi-generic	1008
Архитектура модуля	1008
Компоненты модуля	1009
Взаимодействия модуля	1010
Дополнительные ресурсы	1010
Модуль csi-vsphere	1010
Архитектура модуля	1010
Компоненты модуля	1011
Взаимодействия модуля	1011
Дополнительные ресурсы	1012
Модуль csi-yadro-tatlin-unified	1012
Архитектура модуля	1012
Компоненты модуля	1013
Взаимодействия модуля	1014
Дополнительные ресурсы	1014
CSI-драйвер (типовой)	1014
Архитектура драйвера	1014
Компоненты драйвера	1015
Взаимодействия драйвера	1016
Дополнительные ресурсы	1017
CSI-драйвер Ceph RBD	1017
Архитектура CSI-драйвера	1017
Компоненты драйвера	1018
Взаимодействия драйвера	1019
Дополнительные ресурсы	1019
CSI-драйвер Huawei	1019
Архитектура драйвера	1020
Компоненты драйвера	1020
Взаимодействия драйвера	1021
Дополнительные ресурсы	1022
CSI-драйвер NFS	1022
Архитектура драйвера	1022
Компоненты драйвера	1023
Взаимодействия драйвера	1024
Дополнительные ресурсы	1025
CSI-драйвер S3	1025
Архитектура CSI-драйвера (csi-s3)	1025
Компоненты драйвера	1025
Взаимодействия драйвера	1026
CSI-драйвер SCSI Generic	1026
Архитектура CSI-драйвера (csi-scsi-generic)	1027
Компоненты драйвера	1027
Взаимодействия драйвера	1028
CSI-драйвер VMware vSphere	1029
Архитектура драйвера	1029
Компоненты драйвера	1030
Взаимодействия драйвера	1031
Дополнительные ресурсы	1032
Подсистема Observability	1032
Архитектура мониторинга в Deckhouse Kubernetes Platform	1032
Состав и схема взаимодействия компонентов мониторинга	1032
Компоненты, устанавливаемые DKP	1033

Prometheus	1034
Что в поде с Prometheus?	1034
Как настраивается Prometheus?	1034
Где Prometheus берет список target'ов?	1035
Как обрабатываются Service Monitor'ы?	1036
Как обрабатываются кастомные ресурсы с rule'ами?	1037
Архитектура оценки доступности компонентов DKP (upmeter)	1037
Дополнительные ресурсы	1037
Модуль prometheus	1038
Архитектура модуля	1038
Компоненты модуля	1038
Взаимодействия модуля	1040
Режим отказоустойчивости и высокой доступности мониторинга (HA)	1040
Дополнительные ресурсы	1040
Модуль observability	1041
Архитектура модуля	1041
Компоненты модуля	1041
Взаимодействия модуля	1042
Дополнительные ресурсы	1043
Модуль extended-monitoring	1043
Архитектура модуля	1043
Компоненты модуля	1044
Взаимодействия модуля	1045
Дополнительные ресурсы	1045
Модуль monitoring-kubernetes	1045
Архитектура модуля	1045
Компоненты модуля	1046
Взаимодействия модуля	1047
Дополнительные ресурсы	1047
Модуль upmeter	1047
Архитектура модуля	1047
Компоненты модуля	1047
Взаимодействия модуля	1049
Дополнительные ресурсы	1049
Модули логирования	1049
Модуль log-shipper	1049
Архитектура модуля	1049
Компоненты модуля	1050
Взаимодействия модуля	1050
Модуль loki	1051
Архитектура модуля	1051
Компоненты модуля	1052
Взаимодействия модуля	1052
Дополнительные ресурсы	1052
Подсистема Managed Services	1052
Дополнительные ресурсы	1053
Модуль managed-memcached	1053
Архитектура модуля	1053
Компоненты модуля	1054
Взаимодействия модуля	1054
Дополнительные ресурсы	1054
Модуль managed-postgres	1054
Архитектура модуля	1055
Компоненты модуля	1056
Взаимодействия модуля	1057
Дополнительные ресурсы	1058
Модуль managed-valkey	1058
Архитектура модуля	1058
Компоненты модуля	1058
Взаимодействия модуля	1059
Дополнительные ресурсы	1059
Подсистема Virtualization	1059

Дополнительные ресурсы	1059
Ядро модуля	1059
Архитектура ядра модуля	1060
Компоненты ядра модуля	1061
Взаимодействия ядра модуля	1063
Дополнительные ресурсы	1063
Deckhouse Virtualization Container Registry (DVCR)	1063
Архитектура DVCR	1063
Компоненты DVCR	1064
Взаимодействия DVCR	1064
Дополнительные ресурсы	1065
Containerized Data Importer (CDI)	1065
Архитектура CDI	1065
Компоненты CDI	1066
Взаимодействия CDI	1067
Дополнительные ресурсы	1068
Вспомогательные компоненты	1068
Аудит событий безопасности	1068
Архитектура	1068
Компоненты	1068
Взаимодействия	1069
Virtualization-dra и прочие компоненты	1069
Архитектура	1069
Компоненты	1070
Взаимодействия	1071
Дополнительные ресурсы	1071
Глобальные настройки	1071
Конверсии	1072
Параметры	1072
Дополнительные ресурсы	1076
Custom Resources	1077
Application	1077
ApplicationPackage	1079
ApplicationPackageVersion	1079
ClusterConfiguration	1084
CNIMigration	1088
CNINodeMigration	1089
DeckhouseRelease	1090
InitConfiguration	1091
Module	1092
ModuleConfig	1094
ModuleDocumentation	1095
ModulePullOverride	1095
ModuleRelease	1097
ModuleSettingsDefinition	1098
ModuleSource	1099
ModuleUpdatePolicy	1101
PackageRepository	1105
PackageRepositoryOperation	1107
SSHConfig	1110
SSHHost	1111
StaticClusterConfiguration	1111
Каналы обновлений	1112
Дополнительные ресурсы	1112
Поддерживаемые версии Kubernetes и ОС	1112
Linux	1112
Kubernetes	1113
Container registry	1113
Сравнение редакций Deckhouse Kubernetes Platform	1114
Структура хранилища образов контейнеров	1119
Хранилище образов сертифицированной версии	1119
Хранилище образов коммерческих версий	1119

Сетевое взаимодействие компонентов платформы	1119
Требования к сетевым задержкам между master-узлами	1120
Возможные проблемы	1120
Проверка текущих задержек	1121
Сетевые задержки на остальных узлах	1121
Трафик между master-узлами	1121
Трафик от master-узлов к узлам	1121
Трафик от узлов к master-узлам	1122
Трафик между узлами	1122
Статические порты модуля virtualization на узлах кластера (зарезервирован диапазон 4100-4134/TCP)	1126
Внешний трафик к master-узлам	1127
Внешний трафик к фронтенд-узлам	1127
Внешний трафик для всех узлов	1127
Дополнительные ресурсы	1128
Список алертов	1129
Критичность алерта	1129
Модуль admission-policy-engine	1129
Модуль cert-manager	1130
Модуль chrony	1132
Модуль cloud-provider-huaweicloud	1132
Модуль cloud-provider-yandex	1132
Модуль cni-cilium	1134
Модуль control-plane-manager	1135
Модуль documentation	1138
Модуль extended-monitoring	1139
Модуль ingress-nginx	1150
Модуль istio	1153
Модуль kube-dns	1156
Модуль log-shipper	1156
Модуль loki	1158
Модуль metallb	1158
Модуль monitoring-applications	1159
Модуль monitoring-custom	1159
Модуль monitoring-deckhouse	1160
Модуль monitoring-kubernetes	1168
Модуль monitoring-kubernetes-control-plane	1175
Модуль monitoring-ping	1177
Модуль node-local-dns	1177
Модуль node-manager	1178
Модуль okmeter	1186
Модуль openvpn	1186
Модуль operator-prometheus	1188
Модуль prometheus	1189
Модуль secret-copier	1194
Модуль terraform-manager	1194
Модуль upmeter	1196
Модуль user-authn	1201
Модуль user-authz	1201
Параметры sysctl, настраиваемые платформой	1202
Дополнительные ресурсы	1205
Информация об используемом ПО	1205
Используемые директории	1211
Описание и установка Deckhouse CLI	1211
Основные группы команд	1211
Как установить Deckhouse CLI	1212
Установка с помощью trdl	1212
Установка исполняемого файла	1213
Справка по командам Deckhouse CLI	1217
d8 backup	1217
d8 backup cluster-config	1217
d8 backup etcd	1217

d8 backup loki	1218
d8 cli	1218
d8 cli check	1218
d8 cli update	1219
d8 cli use	1219
d8 cli versions	1220
d8 completion	1220
d8 completion bash	1220
d8 completion fish	1220
d8 completion powershell	1220
d8 completion zsh	1221
d8 cr	1221
d8 cr catalog	1221
d8 cr config	1222
d8 cr digest	1222
d8 cr export	1222
d8 cr fs	1223
d8 cr fs cat	1223
d8 cr fs extract	1224
d8 cr fs ls	1224
d8 cr fs tree	1225
d8 cr login	1225
d8 cr ls	1225
d8 cr manifest	1226
d8 cr pull	1226
d8 cr push	1227
d8 data	1227
d8 data create	1227
d8 data delete	1228
d8 data download	1228
d8 data export	1228
d8 data export create	1228
d8 data export delete	1228
d8 data export download	1229
d8 data export list	1229
d8 data import	1229
d8 data import create	1230
d8 data import delete	1230
d8 data import upload	1230
d8 data list	1230
d8 delivery-kit	1231
d8 delivery-kit build	1231
d8 delivery-kit bundle	1236
d8 delivery-kit bundle apply	1236
d8 delivery-kit bundle copy	1242
d8 delivery-kit bundle plan	1244
d8 delivery-kit bundle publish	1249
d8 delivery-kit bundle render	1257
d8 delivery-kit chart	1262
d8 delivery-kit chart ts	1262
d8 delivery-kit chart ts build	1262
d8 delivery-kit chart ts init	1263
d8 delivery-kit ci-env	1265
d8 delivery-kit cleanup	1266
d8 delivery-kit completion	1271
d8 delivery-kit compose	1271
d8 delivery-kit compose config	1271
d8 delivery-kit compose down	1276
d8 delivery-kit compose run	1281
d8 delivery-kit compose up	1287
d8 delivery-kit config	1292
d8 delivery-kit config graph	1292

d8 delivery-kit config list	1293
d8 delivery-kit config render	1295
d8 delivery-kit converge	1296
d8 delivery-kit cr	1306
d8 delivery-kit cr login	1306
d8 delivery-kit cr logout	1307
d8 delivery-kit dismiss	1308
d8 delivery-kit export	1314
d8 delivery-kit helm	1318
d8 delivery-kit helm create	1319
d8 delivery-kit helm dependency	1320
d8 delivery-kit helm dependency build	1321
d8 delivery-kit helm dependency list	1323
d8 delivery-kit helm dependency update	1324
d8 delivery-kit helm env	1325
d8 delivery-kit helm get	1326
d8 delivery-kit helm get all	1327
d8 delivery-kit helm get hooks	1328
d8 delivery-kit helm get manifest	1329
d8 delivery-kit helm get metadata	1330
d8 delivery-kit helm get notes	1331
d8 delivery-kit helm get values	1332
d8 delivery-kit helm get-autogenerated-values	1333
d8 delivery-kit helm get-namespace	1337
d8 delivery-kit helm get-release	1339
d8 delivery-kit helm history	1341
d8 delivery-kit helm install	1342
d8 delivery-kit helm lint	1345
d8 delivery-kit helm list	1347
d8 delivery-kit helm package	1349
d8 delivery-kit helm plugin	1350
d8 delivery-kit helm plugin install	1351
d8 delivery-kit helm plugin list	1352
d8 delivery-kit helm plugin uninstall	1353
d8 delivery-kit helm plugin update	1354
d8 delivery-kit helm pull	1355
d8 delivery-kit helm registry	1357
d8 delivery-kit helm registry login	1358
d8 delivery-kit helm registry logout	1359
d8 delivery-kit helm repo	1360
d8 delivery-kit helm repo add	1361
d8 delivery-kit helm repo index	1362
d8 delivery-kit helm repo list	1363
d8 delivery-kit helm repo remove	1364
d8 delivery-kit helm repo update	1365
d8 delivery-kit helm rollback	1366
d8 delivery-kit helm search	1368
d8 delivery-kit helm search hub	1369
d8 delivery-kit helm search repo	1370
d8 delivery-kit helm secret	1371
d8 delivery-kit helm secret decrypt	1372
d8 delivery-kit helm secret encrypt	1373
d8 delivery-kit helm secret file	1375
d8 delivery-kit helm secret file decrypt	1376
d8 delivery-kit helm secret file edit	1377
d8 delivery-kit helm secret file encrypt	1378
d8 delivery-kit helm secret generate-secret-key	1380
d8 delivery-kit helm secret rotate-secret-key	1381
d8 delivery-kit helm secret values	1382
d8 delivery-kit helm secret values decrypt	1383
d8 delivery-kit helm secret values edit	1385
d8 delivery-kit helm secret values encrypt	1386

d8 delivery-kit helm show	1387
d8 delivery-kit helm show all	1388
d8 delivery-kit helm show chart	1390
d8 delivery-kit helm show crds	1391
d8 delivery-kit helm show readme	1393
d8 delivery-kit helm show values	1394
d8 delivery-kit helm status	1396
d8 delivery-kit helm template	1397
d8 delivery-kit helm test	1401
d8 delivery-kit helm uninstall	1402
d8 delivery-kit helm upgrade	1403
d8 delivery-kit helm verify	1407
d8 delivery-kit helm version	1408
d8 delivery-kit host	1409
d8 delivery-kit host cleanup	1409
d8 delivery-kit host purge	1411
d8 delivery-kit includes	1412
d8 delivery-kit includes get-file	1412
d8 delivery-kit includes ls-files	1414
d8 delivery-kit includes update	1416
d8 delivery-kit kube-run	1417
d8 delivery-kit lint	1422
d8 delivery-kit managed-images	1430
d8 delivery-kit managed-images add	1430
d8 delivery-kit managed-images ls	1433
d8 delivery-kit managed-images rm	1436
d8 delivery-kit plan	1440
d8 delivery-kit purge	1449
d8 delivery-kit render	1453
d8 delivery-kit rollback	1462
d8 delivery-kit run	1466
d8 delivery-kit sbom	1470
d8 delivery-kit sbom get	1470
d8 delivery-kit sbom merge	1476
d8 delivery-kit sbom validate	1477
d8 delivery-kit slugify	1478
d8 delivery-kit stages	1479
d8 delivery-kit stages copy	1479
d8 delivery-kit synchronization	1483
d8 delivery-kit verify	1485
d8 delivery-kit version	1486
d8 help	1486
d8 iam	1486
d8 iam access	1486
d8 iam access grant	1486
d8 iam access revoke	1487
d8 iam get	1488
d8 iam get group	1488
d8 iam get rule	1488
d8 iam get user	1488
d8 iam group	1489
d8 iam group add-member	1489
d8 iam group create	1489
d8 iam group delete	1490
d8 iam group remove-member	1490
d8 iam list	1490
d8 iam list groups	1490
d8 iam list rules	1491
d8 iam list users	1491
d8 iam user	1491
d8 iam user create	1492
d8 iam user delete	1492

d8 iam user lock	1493
d8 iam user reset-password	1493
d8 iam user reset2fa	1494
d8 iam user unlock	1494
d8 k	1494
d8 k alpha	1495
d8 k annotate	1497
d8 k api-resources	1499
d8 k api-versions	1501
d8 k apply	1502
d8 k apply edit-last-applied	1505
d8 k apply set-last-applied	1507
d8 k apply view-last-applied	1509
d8 k attach	1510
d8 k auth	1512
d8 k auth can-i	1513
d8 k auth reconcile	1515
d8 k auth whoami	1516
d8 k autoscale	1518
d8 k certificate	1520
d8 k certificate approve	1522
d8 k certificate deny	1523
d8 k cluster-info	1525
d8 k cluster-info dump	1527
d8 k completion	1528
d8 k config	1530
d8 k config current-context	1531
d8 k config delete-cluster	1532
d8 k config delete-context	1534
d8 k config delete-user	1535
d8 k config get-clusters	1536
d8 k config get-contexts	1538
d8 k config get-users	1539
d8 k config rename-context	1540
d8 k config set	1542
d8 k config set-cluster	1543
d8 k config set-context	1544
d8 k config set-credentials	1546
d8 k config unset	1548
d8 k config use-context	1549
d8 k config view	1550
d8 k cordon	1552
d8 k cp	1553
d8 k create	1555
d8 k create clusterrole	1557
d8 k create clusterrolebinding	1559
d8 k create configmap	1561
d8 k create cronjob	1563
d8 k create deployment	1565
d8 k create ingress	1567
d8 k create job	1569
d8 k create namespace	1571
d8 k create poddisruptionbudget	1573
d8 k create priorityclass	1575
d8 k create quota	1577
d8 k create role	1579
d8 k create rolebinding	1581
d8 k create secret	1584
d8 k create secret docker-registry	1585
d8 k create secret generic	1587
d8 k create secret tls	1589
d8 k create service	1591

d8 k create service clusterip	1593
d8 k create service externalname	1594
d8 k create service loadbalancer	1596
d8 k create service nodeport	1598
d8 k create serviceaccount	1600
d8 k create token	1602
d8 k debug	1604
d8 k delete	1606
d8 k describe	1609
d8 k diff	1610
d8 k drain	1612
d8 k edit	1614
d8 k events	1616
d8 k exec	1618
d8 k explain	1620
d8 k expose	1621
d8 k get	1624
d8 k kustomize	1626
d8 k label	1628
d8 k logs	1631
d8 k options	1633
d8 k patch	1634
d8 k plugin	1636
d8 k plugin list	1637
d8 k port-forward	1639
d8 k proxy	1640
d8 k replace	1642
d8 k rollout	1644
d8 k rollout history	1646
d8 k rollout pause	1648
d8 k rollout restart	1649
d8 k rollout resume	1651
d8 k rollout status	1653
d8 k rollout undo	1655
d8 k run	1657
d8 k scale	1660
d8 k set	1662
d8 k set env	1663
d8 k set image	1665
d8 k set resources	1667
d8 k set selector	1670
d8 k set serviceaccount	1672
d8 k set subject	1674
d8 k taint	1676
d8 k top	1678
d8 k top node	1679
d8 k top pod	1681
d8 k uncordon	1682
d8 k version	1684
d8 k wait	1685
d8 mirror	1687
d8 mirror pull	1687
d8 mirror push	1691
d8 network	1691
d8 network cni-migration	1691
d8 network cni-migration cleanup	1691
d8 network cni-migration switch	1692
d8 network cni-migration watch	1692
d8 package	1692
d8 package bootstrap	1692
d8 package bootstrap application	1692
d8 package bootstrap module	1692

d8 package build	1693
d8 package render	1693
d8 package verify	1693
d8 snapshot	1694
d8 snapshot create	1694
d8 snapshot delete	1694
d8 snapshot describe	1695
d8 snapshot download	1695
d8 snapshot get	1696
d8 snapshot local	1696
d8 snapshot local describe	1696
d8 snapshot local get	1696
d8 snapshot restore	1697
d8 snapshot upload	1697
d8 status	1698
d8 stronghold	1698
d8 stronghold audit	1698
d8 stronghold audit disable	1698
d8 stronghold audit enable	1698
d8 stronghold audit list	1699
d8 stronghold auth	1699
d8 stronghold auth disable	1699
d8 stronghold auth enable	1699
d8 stronghold auth help	1699
d8 stronghold auth list	1699
d8 stronghold auth move	1699
d8 stronghold auth tune	1699
d8 stronghold debug	1699
d8 stronghold delete	1700
d8 stronghold events	1700
d8 stronghold events subscribe	1700
d8 stronghold kv	1700
d8 stronghold kv delete	1700
d8 stronghold kv destroy	1700
d8 stronghold kv enable-versioning	1700
d8 stronghold kv get	1700
d8 stronghold kv list	1701
d8 stronghold kv metadata	1701
d8 stronghold kv metadata delete	1701
d8 stronghold kv metadata get	1701
d8 stronghold kv metadata patch	1701
d8 stronghold kv metadata put	1701
d8 stronghold kv patch	1701
d8 stronghold kv put	1701
d8 stronghold kv rollback	1701
d8 stronghold kv undelete	1702
d8 stronghold lease	1702
d8 stronghold lease lookup	1702
d8 stronghold lease renew	1702
d8 stronghold lease revoke	1702
d8 stronghold list	1702
d8 stronghold login	1702
d8 stronghold monitor	1702
d8 stronghold namespace	1703
d8 stronghold namespace create	1703
d8 stronghold namespace delete	1703
d8 stronghold namespace list	1703
d8 stronghold namespace lock	1703
d8 stronghold namespace lookup	1703
d8 stronghold namespace patch	1703
d8 stronghold namespace unlock	1703
d8 stronghold operator	1703

d8 stronghold operator generate-root	1704
d8 stronghold operator init	1704
d8 stronghold operator key-status	1704
d8 stronghold operator members	1704
d8 stronghold operator raft	1704
d8 stronghold operator raft autopilot	1704
d8 stronghold operator raft autopilot get-config	1704
d8 stronghold operator raft autopilot set-config	1704
d8 stronghold operator raft autopilot state	1705
d8 stronghold operator raft demote	1705
d8 stronghold operator raft join	1705
d8 stronghold operator raft list-peers	1705
d8 stronghold operator raft promote	1705
d8 stronghold operator raft remove-peer	1705
d8 stronghold operator raft snapshot	1705
d8 stronghold operator raft snapshot restore	1705
d8 stronghold operator raft snapshot save	1705
d8 stronghold operator rekey	1706
d8 stronghold operator rotate	1706
d8 stronghold operator seal	1706
d8 stronghold operator step-down	1706
d8 stronghold operator unseal	1706
d8 stronghold operator usage	1706
d8 stronghold patch	1706
d8 stronghold path-help	1706
d8 stronghold pki	1707
d8 stronghold pki health-check	1707
d8 stronghold pki issue	1707
d8 stronghold pki list-intermediates	1707
d8 stronghold pki reissue	1707
d8 stronghold pki verify-sign	1707
d8 stronghold plugin	1707
d8 stronghold plugin deregister	1707
d8 stronghold plugin info	1707
d8 stronghold plugin list	1708
d8 stronghold plugin register	1708
d8 stronghold plugin reload	1708
d8 stronghold plugin reload-status	1708
d8 stronghold policy	1708
d8 stronghold policy delete	1708
d8 stronghold policy list	1708
d8 stronghold policy read	1708
d8 stronghold policy write	1709
d8 stronghold print	1709
d8 stronghold print token	1709
d8 stronghold read	1709
d8 stronghold secrets	1709
d8 stronghold secrets disable	1709
d8 stronghold secrets enable	1709
d8 stronghold secrets list	1709
d8 stronghold secrets move	1709
d8 stronghold secrets tune	1710
d8 stronghold ssh	1710
d8 stronghold status	1710
d8 stronghold token	1710
d8 stronghold token capabilities	1710
d8 stronghold token create	1710
d8 stronghold token lookup	1710
d8 stronghold token renew	1710
d8 stronghold token revoke	1711
d8 stronghold transit	1711
d8 stronghold transit import	1711

d8 stronghold transit import-version	1711
d8 stronghold unwrap	1711
d8 stronghold version	1711
d8 stronghold version-history	1711
d8 stronghold write	1711
d8 system	1711
d8 system collect-debug-info	1712
d8 system edit	1712
d8 system edit cluster-configuration	1712
d8 system edit provider-cluster-configuration	1713
d8 system edit static-cluster-configuration	1713
d8 system get	1713
d8 system get cluster-configuration	1714
d8 system get provider-cluster-configuration	1714
d8 system get static-cluster-configuration	1714
d8 system logs	1714
d8 system module	1715
d8 system module apply-now	1715
d8 system module approve	1715
d8 system module disable	1715
d8 system module enable	1716
d8 system module list	1716
d8 system module maintenance	1716
d8 system module maintenance disable	1716
d8 system module maintenance enable	1717
d8 system module snapshots	1717
d8 system module values	1717
d8 system package	1717
d8 system package scan	1718
d8 system queue	1718
d8 system queue list	1718
d8 system queue main	1719
d8 tools	1719
d8 tools far-convert	1719
d8 tools gostsum	1719
d8 tools imagedigest	1719
d8 tools imagedigest add	1719
d8 tools imagedigest calculate	1720
d8 tools imagedigest calculate-from-file	1720
d8 tools imagedigest validate	1720
d8 tools pki	1721
d8 tools pki certs	1721
d8 tools pki certs check	1721
d8 tools pki certs renew	1721
d8 tools pki certs renew all	1721
d8 tools sig-migrate	1722
d8 v	1722
d8 v ansible-inventory	1723
d8 v collect-debug-info	1724
d8 v console	1726
d8 v evict	1727
d8 v migrate	1728
d8 v port-forward	1729
d8 v restart	1731
d8 v scp	1732
d8 v ssh	1733
d8 v start	1735
d8 v stop	1736
d8 v vnc	1737
Частые вопросы	1740
Общее	1740
Как принудительно обновить модуль?	1740

Как применить обновление DKP, минуя окна обновлений, canary-release и ручной режим обновлений?	1740
Что делать, если при включенном VPN контейнер с установщиком не может получить доступ к сети?	1740
Как сменить канал обновлений для модуля?	1741
Как сделать так, чтобы внешний модуль обновлялся только на патч-версии?	1742
Что делать, если образ модуля не скачался и модуль не переустановился?	1743
Кластер и инфраструктура	1744
Что делать в случае проблем с добавлением узла в кластер через Cluster API Provider Static?	1744
Как изменить тип инстанса у узлов с типом CloudPermanent?	1744
Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?	1746
Если в кластере с одной группой узлов возникают проблемы с заказом узлов	1747
Пример процесса заказа узлов с настройками по умолчанию	1747
Что делать, если после добавления объекта не создаются порождаемые им ресурсы?	1747
Deckhouse	1749
Как узнать версию DKP на каждом из каналов обновлений?	1749
Как проверить очередь заданий в Deckhouse?	1749
Как посмотреть состояние всех очередей заданий Deckhouse?	1749
Как посмотреть состояние очереди заданий main?	1749
Что делать, если DKP не получает обновления из настроенного канала?	1749
Пример настроек DNS, которые могут привести к ошибкам в разрешении IP-адреса хранилища образов DKP	1750
Поддерживает ли Deckhouse установку на ОС с realtime (rt) и low-latency ядрами?	1750
Как узнать, что для кластера доступна новая версия DKP?	1750
Как заранее получать информацию о предстоящем обновлении?	1751
Как понять, что обновление прошло успешно?	1751
Если что-то пошло не так	1751
Как понять, что в кластере идет обновление?	1751
Что делать при наличии проблем с обновлением DKP?	1752
Обновление Deckhouse Kubernetes Platform не проходит, один или несколько подов Deckhouse в нерабочем состоянии	1752
Обновление DKP находится в статусе Release is suspended	1752
Идентификация и доступ	1753
Что делать при проблемах с применением настроек DexProvider?	1753
Kubernetes и планировщик	1753
Что делать при высокой нагрузке на API-сервер?	1753
Как проверить используемую версию Kubernetes?	1754
Сеть	1754
Поддерживается ли модуль ingress-nginx? Не мешает ли это пройти аудит PCI DSS?	1754
История изменений Deckhouse Kubernetes Platform	1754
Версия 1.76	1754
Обратите внимание	1754
Подсистема Deckhouse	1754
Подсистема Kubernetes & Scheduling	1755
Подсистема IAM	1755
Подсистема Security	1756
Подсистема Cluster & Infrastructure	1756
Подсистема Network	1757
Версия 1.75	1757
Обратите внимание	1757
Подсистема Deckhouse	1758
Подсистема Kubernetes & Scheduling	1758
Подсистема IAM	1758
Подсистема Security	1759
Подсистема Cluster & Infrastructure	1759
Подсистема Network	1759
Версия 1.74	1760
Обратите внимание	1760
Основные изменения	1760
Безопасность	1760
Сеть	1760

Версия 1.73	1761
Обратите внимание	1761
Основные изменения	1761
Безопасность	1762
Сеть	1762
Версия 1.72	1762
Обратите внимание	1762
Основные изменения	1762
Безопасность	1763
Версия 1.71	1763
Обратите внимание	1763
Основные изменения	1763
Безопасность	1764
Сеть	1764
Версия 1.70	1765
Обратите внимание	1765
Основные изменения	1765
Безопасность	1766
Версия 1.69	1766
Обратите внимание	1766
Основные изменения	1766
Безопасность	1767
Версия 1.68	1767
Обратите внимание	1767
Основные изменения	1767
Безопасность	1768
Перезапуск компонентов	1768
Встроенные модули Deckhouse Kubernetes Platform	1769
Модуль admission-policy-engine	1769
Особенности отображения сообщений о неудачной валидации объектов	1769
Валидация подов при изменении политики или добавлении новой	1770
Pod Security Standards	1773
Операционные политики	1774
Политики безопасности	1775
Изменение ресурсов Kubernetes	1777
Настройки	1777
Параметры	1779
Custom resources	1780
OperationPolicy	1781
SecurityPolicyException	1785
SecurityPolicy	1790
Custom Resources (от Gatekeeper)	1799
Mutation Custom Resources	1799
AssignMetadata	1800
Assign	1800
ModifySet	1801
AssignImage	1801
FAQ	1802
Как настроить альтернативные решения по управлению политиками безопасности?	1802
Как настроить селекторы политик?	1802
spec.match.labelSelector – выбор подов	1802
spec.match.namespaceSelector – выбор неймспейсов	1803
Совместное использование селекторов	1804
Как расширить политики Pod Security Standards?	1804
Как включить одну или несколько политик Pod Security Standards, не отключая весь набор?	1805
Что, если несколько политик (операционных или безопасности) применяются на один объект?	1808
Проверка подписи образов	1809
Как запретить удаление узла без лейбла	1809
Как запретить операции kubectl exec и kubectl attach в определённые поды?	1810
Встроенная политика для подов с heritage: deckhouse	1810

Пример пользовательской политики	1811
Модуль cert-manager	1812
Возможности модуля cert-manager (с учетом внесенных изменений)	1812
Мониторинг	1812
Роли доступа к ресурсам	1812
Настройки	1812
Параметры	1814
Custom resources	1819
Примеры конфигурации	1819
Пример заказа сертификата	1819
Заказ wildcard-сертификата с DNS в Cloudflare	1819
Заказ wildcard-сертификата с DNS в Route53	1820
Заказ wildcard-сертификата с DNS в Google	1821
Заказ self-signed-сертификата	1822
FAQ	1822
Какие виды сертификатов поддерживаются?	1822
Как добавить дополнительный ClusterIssuer?	1822
В каких случаях требуется дополнительный ClusterIssuer?	1822
Как добавить дополнительный ClusterIssuer, использующий Let's Encrypt и метод подтверждения DNS-01?	1823
Как добавить дополнительный Issuer и ClusterIssuer, использующий HashiCorp Vault для выпуска сертификатов?	1824
Как добавить ClusterIssuer, использующий свой или промежуточный CA для заказа сертификатов?	1825
Как защитить учетные данные cert-manager?	1826
Работает ли аннотация TLS-acme?	1827
Как посмотреть состояние сертификата?	1828
Как получить список сертификатов?	1829
Что делать, если появляется ошибка: CAA record does not match issuer?	1829
Модуль chrony	1829
Как работает	1829
Настройки	1830
Параметры	1832
Пример конфигурации	1832
FAQ	1832
Как запретить использование chrony и использовать NTP-демоны на узлах?	1832
Модуль cilium-hubble	1832
Требования	1832
Настройки	1833
Требования	1834
Конверсии	1834
Параметры	1835
Аутентификация	1837
Cloud provider — AWS	1837
Настройки	1838
Параметры	1839
Настройки провайдера	1840
AWSClusterConfiguration	1840
Подготовка окружения	1848
JSON-спецификация Policy	1849
Настройка IAM через веб-интерфейс	1851
Настройка IAM через CLI	1852
Настройка IAM через Terraform	1853
Custom Resources	1853
AWSInstanceClass	1853
FAQ	1857
Как поднять пиринговое соединение между VPC?	1857
Как создать кластер в новом VPC с доступом через имеющийся bastion-хост?	1857
Как создать кластер в новом VPC и развернуть bastion-хост для доступа к узлам?	1858
Особенности настройки bastion	1858
Добавление CloudStatic-узлов в кластер	1858
Как увеличить размер volume в кластере?	1858

Как настроить доступ к репозиторию Amazon ECR на узлах кластера	1858
Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?	1859
Примеры	1859
Пример custom resource AWSInstanceClass	1859
LoadBalancer	1859
Аннотации объекта Service	1859
Настройка политик безопасности на узлах	1860
Установка дополнительных security groups на статических и master-узлах	1860
Установка дополнительных security groups на эфемерных узлах	1860
Настройка балансировщика в случае наличия Ingress-узлов не во всех зонах	1860
Схемы размещения	1860
WithoutNAT	1860
WithNAT	1862
Cloud provider — Azure	1864
Настройки	1865
Параметры	1867
Настройки провайдера	1868
AzureClusterConfiguration	1868
Подготовка окружения	1875
Custom Resources	1877
AzureInstanceClass	1877
Примеры	1880
Пример custom resource AzureInstanceClass	1880
Схемы размещения	1880
Standard	1880
Cloud provider — DVP	1880
Настройки	1881
Параметры	1882
Настройки провайдера	1883
DVPClusterConfiguration	1883
Подготовка окружения	1893
Создание пользователя	1894
Добавление роли	1894
Генерация kubeconfig	1894
Custom Resources	1895
DVPIInstanceClass	1895
Схемы размещения	1898
Standard	1898
Cloud provider — Basis Dynamix	1899
Настройки	1899
Настройки провайдера	1900
DynamixClusterConfiguration	1901
Подготовка окружения.	1906
Подготовка образа операционной системы	1906
Custom Resources	1906
DynamixInstanceClass	1906
FAQ	1907
Как настроить LoadBalancer?	1907
Cloud provider - Basis Dynamix: схемы размещения	1907
Standard	1907
StandardWithInternalNetwork	1908
Cloud provider — GCP	1909
Настройки	1909
Параметры	1911
Настройки провайдера	1911
GCPClusterConfiguration	1911
Подготовка окружения	1916
Настройка через Google Cloud Console	1917
Настройка через gcloud CLI	1917
Custom Resources	1918
GCPIInstanceClass	1918

FAQ	1923
Как поднять кластер	1923
Добавление CloudStatic узлов в кластер	1923
Примеры	1923
Пример кастомного ресурса GCPInstanceClass	1923
Включение вложенной виртуализации	1924
Добавление дополнительных дисков	1924
Настройка политик безопасности на узлах	1924
Установка дополнительных network tags на статических и master-узлах	1924
Установка дополнительных network tags на эфемерных узлах	1924
Схемы размещения	1924
Standard	1924
WithoutNAT	1926
Cloud provider — Huawei Cloud	1928
Настройки	1929
Настройки провайдера	1929
HuaweiCloudClusterConfiguration	1929
Подготовка окружения	1934
Настройка IAM через веб-интерфейс	1935
JSON-политики	1935
Custom Resources	1937
HuaweiCloudInstanceClass	1937
Схемы размещения	1939
Standard	1939
VpcPeering	1940
Cloud provider — OpenStack	1941
Настройки	1942
Список необходимых сервисов OpenStack	1943
Параметры	1944
Настройки провайдера	1946
OpenStackClusterConfiguration	1946
Подготовка окружения	1956
Custom Resources	1956
OpenStackInstanceClass	1956
FAQ	1960
Как настроить LoadBalancer?	1960
Пример IngressNginxController	1960
Как настроить политики безопасности на узлах кластера?	1961
Установка дополнительных групп безопасности (security groups) на статических и master-узлах	1961
Установка дополнительных групп безопасности (security groups) на ephemeral-узлах	1961
Как создать NodeGroup в зонах доступности?	1961
Как поднять гибридный кластер?	1962
Подключение storage в гибридном кластере	1962
Как загрузить образ в OpenStack?	1962
Как проверить, поддерживает ли провайдер группы безопасности (security groups)?	1964
Как настроить работу онлайн-изменения размера дисков?	1964
Как использовать rootDiskSize и когда он предпочтителен?	1964
Диски в OpenStack	1964
Параметр rootDiskSize	1964
Рекомендация для master-узлов и бастiona — сетевой диск	1964
Рекомендация для ephemeral-узлов — локальный диск	1964
Как проверить объем диска в flavor?	1965
Как переопределить тип диска по умолчанию cloud-провайдера?	1965
Оффлайн-изменение размера диска	1965
Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?	1965
Примеры	1965
Пример 1	1965
Пример 2	1965
Схемы размещения	1966
Standard	1966

StandardWithNoRouter	1968
Simple	1970
SimpleWithInternalNetwork	1972
Cloud provider — VMware Cloud Director	1974
Настройки	1975
Параметры	1976
Настройки провайдера	1977
VCDClusterConfiguration	1977
Подготовка окружения.	1986
Список необходимых ресурсов VCD	1986
Права пользователя	1986
Добавление сети	1989
Настройка DHCP	1992
Добавление vApp	1994
Добавление сети к vApp	1995
Входящий трафик	1997
Настройка правил DNAT/SNAT на edge gateway	1997
Выпуск сертификатов Let's Encrypt	1998
Настройка firewall	1998
Шаблон виртуальной машины	2000
Подготовка шаблона из OVA-файла	2001
Настройка шаблона в VCD	2006
Использование хранилища	2011
Использование балансировщика нагрузки	2011
Custom Resources	2011
VCDInstanceClass	2012
Примеры	2013
Пример конфигурации кастомного ресурса VCDInstanceClass	2013
Cloud provider - VMware Cloud Director: схемы размещения	2013
Standard	2014
WithNAT	2014
Cloud provider — VMware vSphere	2017
Настройки	2017
Конверсии	2019
Storage	2019
CSI	2019
Важная информация об увеличении размера PVC	2019
Параметры	2019
Настройки провайдера	2024
VsphereClusterConfiguration	2024
Подготовка окружения	2036
Требования к окружению	2036
Список необходимых ресурсов vSphere	2036
Список необходимых привилегий	2036
Конфигурация vSphere	2039
Настройка через vSphere Client	2039
Создание тегов и категорий тегов с использованием vSphere Client	2039
Настройка Datastore с использованием vSphere Client	2042
Создание и назначение роли с использованием vSphere Client	2043
Настройка через govc	2044
Установка govc	2044
Создание тегов и категорий тегов с использованием govc	2045
Настройка Datastore с использованием govc	2045
Создание и назначение роли с использованием govc	2045
Требования к образу виртуальной машины	2046
Подготовка образа виртуальной машины	2046
Инфраструктура	2048
Сети	2048
Входящий трафик	2048
Использование хранилища данных	2048
Custom Resources	2048
VsphereInstanceClass	2048

Примеры	2052
Пример конфигурации	2052
Пример custom resource VsphereInstanceClass	2053
Cloud provider - VMware vSphere: схемы размещения	2053
Standard	2053
Cloud provider — Yandex Cloud	2054
Настройки	2054
Storage	2056
LoadBalancer	2056
Параметры	2056
Настройки провайдера	2057
YandexClusterConfiguration	2057
Подготовка окружения	2066
Создание сервисного аккаунта	2066
Квоты	2067
Интеграция с Yandex Cloud	2067
Настройка групп безопасности	2067
Интеграция с Yandex Lockbox	2068
Инструкция по разворачиванию	2068
Проверка работоспособности	2069
Интеграция с Yandex Managed Service for Prometheus	2070
Запись метрик	2070
Чтение метрик через Grafana	2071
Custom Resources	2071
YandexInstanceClass	2071
FAQ	2076
Настройка INTERNAL LoadBalancer	2076
Использование отдельной целевой группы для NodeGroup	2076
Резервирование публичного IP-адреса	2077
Проблемы dhcpOptions и пути их решения	2077
Изменение параметров	2077
Особенности использования	2077
Изменение networkType не пересоздаёт CloudEphemeral-узлы	2078
Назначение произвольного StorageClass используемого по умолчанию	2078
Изменение размера PVC	2078
Добавление CloudStatic-узлов в кластер	2080
Создание кластера в новом VPC и разворачивание bastion-хост для доступа к узлам	2081
Длительное переключение на заказ узлов в менее приоритетных группах	2081
Примеры	2081
Пример конфигурации модуля	2081
Пример custom resource YandexInstanceClass	2081
LoadBalancer	2082
Аннотации объекта Service	2082
Проверки состояния целевой группы	2082
Схемы размещения	2082
Standard	2082
WithoutNAT	2084
WithNATInstance	2086
Cloud provider — zVirt	2088
Настройки	2089
Настройки провайдера	2090
ZvirtClusterConfiguration	2090
Подготовка окружения.	2097
Требования к окружению	2097
Подготовка образа операционной системы	2097
Подготовка шаблона виртуальной машины	2098
Custom Resources	2101
ZvirtInstanceClass	2101
FAQ	2102
Как получить vNicProfileId	2103
Cloud provider - zVirt: схемы размещения	2103
Standard	2103

Модуль cni-cilium	2104
Ограничения	2104
Обработка внешнего трафика в разных режимах работы bpflB (замена kube-proxy от Cilium)	2104
Использование CiliumClusterwideNetworkPolicies	2105
Смена режима работы Cilium	2106
Выключение модуля kube-proxy	2106
Использование выборочного алгоритма балансировки нагрузки для сервисов	2106
Использование Egress Gateway	2106
Базовый режим	2106
Режим с Virtual IP	2106
Экспорт данных из Hubble	2106
Настройки	2107
Параметры	2108
Custom Resources	2113
EgressGatewayPolicy	2113
EgressGateway	2115
HubbleMonitoringConfig	2117
Примеры	2129
Egress Gateway	2129
Принцип работы	2129
Обслуживание узла	2129
Сравнение с CiliumEgressGatewayPolicy	2130
Примеры настроек Egress Gateway	2130
EgressGateway в режиме PrimaryIPFromEgressGatewayNodeInterface (базовый режим)	2130
EgressGateway в режиме VirtualIPAddress (режим с Virtual IP)	2130
EgressGatewayPolicy	2130
HubbleMonitoringConfig	2131
Примеры настроек HubbleMonitoringConfig	2131
Включение расширенных метрик и экспорта flow logs (с фильтрами и маской полей)	2131
Сбор Hubble flow logs с помощью модуля log-shipper	2131
Модуль cni-flannel	2132
Настройки	2132
Параметры	2133
Модуль cni-simple-bridge	2133
Управление control plane	2134
Управление сертификатами	2134
Масштабирование	2134
Масштабирование master-узлов	2135
Динамическое пороговое значение удаления выселенных подов	2135
Управление версиями	2135
Аудит	2135
Feature Gates	2136
Kubernetes 1.31	2136
Kubernetes 1.32	2136
Kubernetes 1.33	2137
Kubernetes 1.34	2137
Kubernetes 1.35	2137
Защита чувствительных полей кастомных ресурсов	2138
Настройки	2138
Конверсии	2140
Параметры	2140
Custom Resources	2145
KubeSchedulerWebhookConfiguration	2146
FAQ	2148
Как добавить master-узел в статическом или гибридном кластере?	2148
Как добавить master-узлы в облачном кластере?	2148
Как уменьшить число master-узлов в облачном кластере?	2149
Как убрать роль master-узла, сохранив узел?	2150
Как изменить образ ОС в кластере с несколькими master-узлами?	2151

В облачном кластере	2151
В статическом кластере	2152
Как изменить образ ОС в кластере с одним master-узлом?	2153
В облачном кластере	2153
В статическом кластере	2153
Как настроить режим HA с двумя master-узлами и arbiter-узлом?	2153
Настройка в облачном кластере	2154
Настройка в статическом кластере	2155
Как посмотреть список узлов кластера в etcd?	2155
Вариант 1	2155
Вариант 2	2155
Что делать, если что-то пошло не так?	2156
Что делать, если кластер etcd не функционирует?	2156
Что делать, если etcd постоянно перезапускается с ошибкой?	2156
Что делать, если объем базы данных etcd достиг лимита, установленного в quota-backend-bytes?	2156
Как сжать базу данных etcd	2157
Как сжать базу данных etcd на узле в кластере с одним master-узлом	2157
Как сжать базу данных etcd в кластере с несколькими master-узлами	2158
Модель административного доступа к кластеру	2159
Файлы kubeconfig на master-узлах	2159
Административный доступ на основе RBAC	2159
Рекомендуемый административный доступ	2159
Символическая ссылка root kubeconfig	2159
Усиление безопасности	2159
Аварийный доступ (break-glass)	2160
Как настроить дополнительные политики аудита?	2160
Как исключить встроенные политики аудита?	2160
Как вывести аудит-лог в стандартный вывод вместо файлов?	2160
Как работать с журналом аудита?	2161
Как ускорить перезапуск подов при потере связи с узлом?	2161
Резервное копирование и восстановление etcd	2161
Что выполняется автоматически	2161
Как сделать резервную копию etcd вручную	2162
Используя Deckhouse CLI (Deckhouse Kubernetes Platform v1.65+)	2162
Используя bash (Deckhouse Kubernetes Platform v1.64 и старше)	2162
Как выполнить полное восстановление состояния кластера из резервной копии etcd?	2162
Восстановление кластера с одним master-узлом	2162
Восстановление кластера с несколькими master-узлами	2163
Как восстановить объект Kubernetes из резервной копии etcd?	2163
Пример шагов по восстановлению объектов из резервной копии etcd	2163
Как выбирается узел, на котором будет запущен под?	2165
Общее устройство планировщика Kubernetes	2165
Логика работы	2165
Профили планировщика	2165
Этапы планирования подов	2165
Документация	2166
Как изменить или расширить логику работы планировщика	2166
Как происходит ротация сертификатов kubelet?	2166
Время жизни сертификатов	2166
Особенности работы с серверными сертификатами kubelet в Deckhouse Kubernetes Platform	2166
Как вручную обновить сертификаты компонентов управляющего слоя?	2167
Как защитить чувствительные поля кастомных ресурсов?	2167
Применяемые меры защиты	2167
Примеры	2167
Подключение внешнего плагина планировщика	2167
Защита ресурсов с чувствительными полями	2168
Модуль csi-vsphere	2169
Основные возможности	2170
Системные требования	2170
Архитектура	2170

Режимы совместимости	2170
Обнаружение и предоставление хранения	2170
Настройка StorageClass по умолчанию	2170
Жизненный цикл томов	2171
Ограничения	2171
Настройки	2171
Параметры	2172
Подготовка окружения	2174
Список необходимых ресурсов	2174
Установка govc	2174
Создание тегов и категорий тегов	2174
Конфигурация Datastore	2174
Создание и назначение роли	2174
Модуль deckhouse	2175
Обновление релизов Deckhouse	2175
Просмотр статуса релизов Deckhouse	2175
Процесс обновления	2175
Ручное применение релизов	2175
Закрепление релиза	2175
Priority Classes	2176
Настройки	2177
Параметры	2178
Custom Resources	2188
ConversionWebhook	2188
ValidationWebhook	2190
Примеры конфигурации	2195
Настройка режима обновления	2195
Конфигурация окон обновлений	2195
Ручное подтверждение обновлений	2195
Оповещение об обновлении Deckhouse	2196
Резервирование имён хостов веб-интерфейсов платформы	2197
Состав резервирования	2198
Исключение из резервирования	2198
Исключение для общего Gateway	2198
Резервирование при наличии работающих приложений в кластере	2198
Проверка состояния резервирования	2199
Сбор информации для отладки	2199
FAQ	2199
Как запустить kube-bench в кластере?	2199
Как собрать информацию для отладки?	2200
Как отлаживать проблемы в подах с помощью ephemeral containers?	2201
Как отлаживать проблемы на узлах с помощью ephemeral containers?	2201
Модуль deckhouse-tools	2201
Настройки	2201
Пример конфигурации	2201
Параметры	2203
Модуль descheduler	2205
Особенности работы модуля	2205
Поставщик метрик	2205
Вытеснение в фоновом режиме	2205
Стратегии	2205
HighNodeUtilization	2205
LowNodeUtilization	2206
RemoveDuplicates	2206
RemovePodsHavingTooManyRestarts	2206
RemovePodsViolatingInterPodAntiAffinity	2206
RemovePodsViolatingNodeAffinity	2207
RemovePodsViolatingTopologySpreadConstraint	2207
Настройки	2207
Конверсии	2208
Параметры	2208
Custom Resources	2208

Descheduler	2209
Примеры	2217
Настройка интервала перераспределения подов	2217
Пример стратегии LowNodeUtilization	2217
Пример стратегии HighNodeUtilization	2218
Модуль documentation	2218
Настройки	2218
Параметры	2219
Аутентификация	2221
Примеры	2222
Пример конфигурации модуля	2222
Модуль extended-monitoring	2222
Настройки	2222
Как использовать extended-monitoring-exporter	2222
Стандартные лейблы и поддерживаемые Kubernetes-объекты	2222
Non-namespaced Kubernetes-объекты	2223
Узел	2223
Namespaced Kubernetes-объекты	2223
Под	2223
Ingress	2223
Deployment	2223
StatefulSet	2223
DaemonSet	2223
CronJob	2224
Как работает	2224
Требования	2225
Конверсии	2225
Параметры	2225
Extended monitoring модуль: FAQ	2227
Как переключиться на HTTP вместо HTTPS для проверки образов из собственного registry?	2227
Модуль ingress-nginx	2227
Маршрутизация трафика	2227
Поддерживаемые протоколы	2228
Использование HTTPS	2228
Мониторинг и статистика	2228
Статистика	2228
Основные принципы сбора статистики	2228
Какая статистика собирается и как она представлена	2228
Поддержка и безопасность модуля	2229
Настройки	2229
Параметры	2230
Custom Resources	2232
IngressNginxController	2232
FAQ	2254
Как разрешить доступ к приложению внутри кластера только от Ingress-контроллера?	2254
Как выдать минимальные права для ServiceAccount?	2255
Конфигурация Kube-RBAC-Proxy	2256
Как сконфигурировать балансировщик нагрузки для проверки доступности IngressNginxController?	2257
Как настроить работу через MetalLB с доступом только из внутренней сети?	2257
Как добавить дополнительные поля для логирования в nginx-controller?	2257
Как включить HorizontalPodAutoscaling для IngressNginxController?	2258
Как использовать IngressClass с установленными IngressClassParameters?	2258
Как отключить сборку детализированной статистики Ingress-ресурсов?	2258
Как корректно вывести из эксплуатации (drain) узел с запущенным IngressNginxController?	2259
Как включить Web Application Firewall (WAF)?	2259
Включение ModSecurity	2259
Настройка ModSecurity	2259
Добавление SecRule	2259
Включение OWASP Core Rule Set (CRS)	2260

Примеры	2260
Пример для AWS (Network Load Balancer)	2260
Пример для GCP / Yandex Cloud / Azure	2260
Пример для OpenStack	2261
Пример для bare metal	2261
Пример для bare metal (при использовании внешнего балансировщика, например Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp и др.)	2261
Пример для bare metal (балансировщик MetalLB в режиме BGP LoadBalancer)	2261
Пример для bare metal (балансировщик MetalLB в режиме L2 LoadBalancer)	2262
Пример разделения доступа между публичной и административной зонами	2263
Особенности использования одного Ingress-контроллера	2263
Использование отдельных Ingress-контроллеров	2264
Модуль istio	2266
Таблица совместимости поддерживаемых версий	2266
Задачи, которые решает Istio	2266
Mutual TLS	2266
Авторизация	2266
Маршрутизация запросов	2267
Управление балансировкой запросов между эндпоинтами сервиса	2267
Observability	2267
Трассировка	2267
Grafana	2267
Kiali	2267
Архитектура кластера с включенным Istio	2267
Архитектура прикладного сервиса с включенным Istio	2269
Особенности	2269
Жизненный цикл пользовательского запроса	2269
Приложение с выключенным Istio	2269
Приложение с включенным Istio	2269
Как активировать Istio для приложения	2269
Федерация и мультикластер	2270
Федерация	2270
Требования к кластерам	2270
Общие принципы федерации	2270
Включение федерации	2271
Управление федерацией	2271
Мультикластер	2271
Требования к кластерам	2271
Общие принципы	2271
Включение мультикластера	2271
Управление мультикластером	2272
Накладные расходы	2272
control-plane	2272
data-plane	2272
Настройки	2272
Конверсии	2274
Параметры	2274
Аутентификация	2291
Кастомные ресурсы	2291
IngressIstioController	2292
IstioFederation	2294
IstioMulticluster	2296
Кастомные ресурсы (от istio.io)	2297
Маршрутизация	2297
DestinationRule	2297
VirtualService	2297
ServiceEntry	2297
Аутентификация	2297
PeerAuthentication	2297
RequestAuthentication	2298
Авторизация	2298
AuthorizationPolicy	2298

Sidecar	2298
Примеры	2298
Circuit Breaker	2298
Балансировка gRPC	2299
Locality Failover	2299
Retry	2300
Canary	2300
Распределение по наличию cookie	2301
Распределение по вероятности	2301
Ingress для публикации приложений	2301
Istio Ingress Gateway	2301
Ingress NGINX	2303
Примеры настройки авторизации	2303
Алгоритм принятия решения	2303
Запретить все действия в рамках неймспейса foo	2305
Запретить доступ только из неймспейса foo	2305
Разрешить запросы только в рамках нашего неймспейса foo	2305
Разрешить из любого места в нашем кластере	2305
Разрешить любые запросы только кластеров foo или bar	2306
Разрешить любые запросы только от сущностей из неймспейса baz кластеров foo или bar	2306
Разрешить из любого кластера (по mTLS)	2306
Разрешить вообще откуда угодно (в том числе без mTLS)	2306
Устройство федерации из двух кластеров с помощью кастомного ресурса IstioFederation	2307
Устройство мультикластера из двух кластеров с помощью ресурса IstioMulticluster	2307
Управление поведением data plane	2307
Предотвращение завершения работы istio-proxy до завершения соединений основного приложения	2307
Ограничения режима перенаправления прикладного трафика CNIPugin	2308
Обновление Istio	2308
Обновление control plane Istio	2308
Автоматическое обновление data plane Istio	2308
Настройка ресурсов istio-proxy sidecar	2309
Поддерживаемые аннотации	2309
Примеры конфигурации	2309
Модуль keepalived	2309
Настройки	2310
Custom resources	2311
KeepalivedInstance	2311
keepalived: FAQ	2314
Как вручную переключить keepalived?	2314
Примеры	2314
Три публичных IP-адреса	2314
Модуль kube-dns	2315
Настройки	2315
Параметры	2317
FAQ	2318
Как поменять домен кластера с минимальным простоем?	2318
Как увеличить количество подов kube-dns?	2320
Примеры	2320
Пример конфигурации	2320
Модуль kube-proxy	2320
Модуль local-path-provisioner	2320
Как это работает?	2320
Ограничения	2321
Настройки	2321
Custom resources	2322
LocalPathProvisioner	2322
FAQ	2323
Как настроить Prometheus на использование локального хранилища?	2323
Примеры	2323

Пример custom resource LocalPathProvisioner	2323
Пример custom resource LocalPathProvisioner с установленным reclaimPolicy	2323
Модуль log-shipper	2324
Топологии отправки	2324
Распределенная	2324
Централизованная	2324
Потоковая	2324
Метаданные	2324
Kubernetes	2324
File	2325
Фильтры сообщений	2325
Настройки	2325
Конверсии	2327
Параметры	2327
Custom Resources	2330
ClusterLogDestination	2330
ClusterLoggingConfig	2358
PodLoggingConfig	2367
The log-shipper module: FAQ	2370
Как добавить авторизацию в ресурс ClusterLogDestination?	2370
Расширенная конфигурация	2371
Как включить debug-логи?	2371
Как узнать больше о каналах передачи log'ов?	2371
Посмотреть каналы как graph	2371
Отладка передачи данных	2372
Получить необработанные образцы log'ов	2372
Как добавить поддержку нового source/sink для log-shipper?	2372
Примеры	2372
Чтение логов из всех подов кластера и направление их в Loki	2372
Чтение логов подов из указанного namespace с указанным label и перенаправление одновременно в Loki и Elasticsearch	2372
Создание source в namespace и чтение логов всех подов в этом NS с направлением их в Loki	2373
Чтение только подов в указанном namespace и с определенным label	2373
Переход с Promtail на Log-Shipper	2374
Работа с Grafana Cloud	2374
Добавление Loki в Deckhouse Grafana	2374
Поддержка Elasticsearch < 6.X	2375
Шаблон индекса для Elasticsearch	2375
Пример интеграции со Splunk	2375
Простой пример Logstash	2376
Syslog	2376
Пример интеграции с Graylog	2376
Логи в CEF формате	2377
Сбор событий Kubernetes	2377
Фильтрация логов	2378
Сборка логов только для контейнера nginx	2378
Сборка логов без строки, содержащей GET /status" 200	2378
Аудит событий kubelet'a	2378
Системные логи Deckhouse	2379
Преобразование логов	2379
Преобразование логов в структурированный объект	2379
Преобразование логов в формате Klog в структурированный объект	2380
Преобразование логов в формате Syslog в структурированный объект	2380
Преобразование логов в формате CLF в структурированный объект	2381
Преобразование логов в формате logfmt в структурированный объект	2381
Парсинг JSON и уменьшение вложенности	2382
Пример парсинга записей смешанных форматов в объект	2382
Замена лейблов	2383
Замена подстрок в строковых полях	2383
Добавление лейблов из литералов и шаблонов	2384
Удаление лейблов	2384

Пример удаления заданного лейбла из структурированного сообщения	2385
Отправка логов в Elasticsearch с трансформациями	2385
Настройка сборки логов с продуктовых неймспейсов, используя опцию namespace label selector	2386
Исключение подов и неймспейсов, используя label	2386
Включение буферизации	2387
Пример включения буферизации в оперативной памяти	2387
Пример включения буферизации на диске	2387
Пример определения поведения при переполнении буфера	2387
Модуль loki	2388
Настройки	2388
Параметры	2389
Пример конфигурации	2394
FAQ	2394
Почему срабатывает алерт LokiDiscardedSamplesWarning?	2394
Возможные причины	2394
Устранение проблемы	2394
Примеры	2394
Чтение логов из всех подов из указанного namespace и направление их в Loki	2394
Модуль metallb	2395
Режим Layer 2	2395
Преимущества модуля перед классическим MetalLB	2395
Режим BGP	2395
Принцип работы metallb в режиме BGP	2396
Преимущества использования BGP	2396
Недостатки использования BGP	2396
Настройки	2396
Конверсии	2398
Параметры	2398
Custom resources	2401
MetalLoadBalancerClass	2401
The metallb module: примеры	2402
Пример использования metallb в режиме L2 LoadBalancer	2402
Пример использования metallb в режиме BGP LoadBalancer	2403
Дополнительные примеры настроек для Service	2404
Создание сервиса с присвоением ему определенных IP-адресов из пула	2404
Создание сервиса с одним принудительно выбранным адресом	2404
Создание сервисов с общими IP адресами	2405
Создание сервиса и назначение ему IPAddressPools при использовании metallb в режиме BGP LoadBalancer	2405
Автоматическая настройка системы мониторинга для сбора метрик с пользовательских приложений	2406
Пример: Service	2407
Пример: Deployment	2407
Настройки	2407
Требования	2409
Модуль monitoring-deckhouse	2409
Сбор метрик	2409
Интеграция с модулем Observability	2409
Требования	2410
Настройки	2410
Требования	2411
Модуль monitoring-kubernetes	2411
Настройки	2412
Требования	2413
Конверсии	2413
Параметры	2413
Мониторинг control plane	2415
Настройки	2415
Требования	2416
Мониторинг сетевого взаимодействия между всеми узлами кластера, а также (опционально) до дополнительных внешних узлов	2416
Описание	2416

Как работает	2417
Модуль monitoring-ping: настройки	2417
Требования	2418
Параметры	2418
Модуль monitoring-ping: примеры конфигурации	2419
Добавление дополнительных IP-адресов для мониторинга	2419
Модуль multitenancy-manager	2419
Описание	2419
Для чего это нужно?	2419
Преимущества модуля	2419
Ограничения	2420
Внутренняя логика работы	2420
Создание проекта	2420
Изоляция проекта	2420
Настройки	2420
Параметры	2422
Custom Resources	2422
Project	2422
ProjectTemplate	2423
Примеры использования	2424
Шаблоны для проектов доступные по умолчанию	2424
Создание проекта	2425
Автоматическое создание проекта для пространства имён	2425
Создание собственного шаблона для проекта	2425
Использование лейблов для управления ресурсами	2426
Пропуск создания лейбла heritage: multitenancy-manager	2426
Исключение ресурсов из управления multitenancy-manager	2426
Реализация валидации изменений объектов с помощью пользовательского лейбла	2427
Как работает валидация в multitenancy-manager	2427
Создание собственной валидации	2427
Модуль namespace-configurator	2428
Как работает	2428
Что нужно настроить?	2428
Настройки	2428
Параметры	2429
Примеры	2430
Пример	2430
Модуль network-gateway	2430
Настройки	2431
Параметры	2432
Модуль network-policy-engine	2433
Настройки	2434
Примеры	2435
Примеры network policies	2435
Запретить обращаться снаружи к подам внутри namespace, но разрешить им обращаться внутри namespace и к внешним ресурсам	2435
Модуль node-local-dns	2436
Описание модуля	2436
Назначение	2436
Grafana dashboard	2436
Как работает	2436
Особенности конфигурации CoreDNS	2436
Настройки	2436
Требования	2438
Параметры	2438
Примеры	2438
Пример настройки кастомного DNS в поде	2438
Пример конфигурации	2439
Управление узлами	2439
Основные функции	2439
Типы узлов	2440
Группировка узлов и управление группами	2440

Задержка перезагрузки или выключения узла, на котором работают критичные поды	2440
Принцип работы и особенности	2440
Особенности механизма в DKP	2440
Автоматическое развертывание, настройка и обновление узлов Kubernetes	2441
Развертывание узлов Kubernetes	2441
Поддержка актуального состояния узлов	2441
Работа с узлами в поддерживаемых облаках	2441
Масштабирование узлов в облаке	2441
Работа со статическими узлами	2441
Cluster API Provider Static	2442
Пользовательские настройки на узлах	2443
Особенности написания скриптов	2445
Chaos Monkey	2445
Мониторинг	2445
Какую информацию собирает Prometheus?	2445
Настройки	2446
Конверсии	2447
Параметры	2447
Custom resources	2449
Instance	2449
InstanceTypesCatalog	2450
NodeGroup	2451
NodeGroupConfiguration	2487
NodeUser	2488
SSHCredentials	2492
StaticInstance	2494
FAQ	2501
Как добавить master-узлы в облачном кластере?	2501
Как уменьшить число master-узлов в облачном кластере?	2501
Статические узлы	2501
Как добавить статический узел в кластер (Cluster API Provider Static)?	2501
Как добавить несколько статических узлов в кластер вручную?	2502
Как вручную очистить статический узел?	2503
Можно ли удалить StaticInstance?	2504
Как изменить IP-адрес StaticInstance?	2504
Как мигрировать статический узел настроенный вручную под управление CAPS?	2504
Почему SSH-ключ и sudo-пароль в SSHCredentials отображаются как <omitted>?	2504
Как изменить NodeGroup у статического узла?	2505
Как очистить узел для последующего ввода в кластер?	2505
Для всех узлов кроме control-plane	2505
Для узлов control-plane	2505
Как понять, что что-то пошло не так?	2506
Как посмотреть, что в данный момент выполняется на узле при его создании?	2506
Как обновить ядро на узлах?	2507
Для дистрибутивов, основанных на Debian	2507
Для дистрибутивов, основанных на CentOS	2507
На что влияют параметры NodeGroup?	2508
Как пересоздать эфемерные машины в облаке с новой конфигурацией?	2509
Как выделить узлы под специфические нагрузки?	2509
Как выделить узлы под системные компоненты?	2509
Фронтенд	2509
Системные	2510
Как ускорить заказ узлов в облаке при горизонтальном масштабировании приложений?	2510
Как выключить machine-controller-manager/CAPi в случае выполнения потенциально деструктивных изменений в кластере?	2510
Как восстановить master-узел, если kubelet не может загрузить компоненты control plane?	2511
containerd	2511
Как изменить CRI для NodeGroup?	2511
Почему изменение CRI могло не примениться?	2512
Как изменить CRI для всего кластера?	2513

Как добавить шаг для конфигурации узлов?	2513
Как автоматически проставить на узел кастомные лейблы?	2513
Как развернуть кастомный конфигурационный файл containerd?	2514
Как добавить конфигурацию для дополнительного registry?	2516
Устаревший способ	2517
Как добавить авторизацию в дополнительный registry (устаревший способ)?	2517
Как настроить сертификат для дополнительного registry (устаревший способ)?	2518
Как добавить TLS skip verify (устаревший способ)?	2519
Как настроить зеркало для доступа к публичным registries (устаревший способ)?	2520
Новый способ	2521
Как добавить авторизацию в дополнительный registry (актуальный способ)?	2522
Как настроить сертификат для дополнительного registry (актуальный способ)?	2523
Как добавить TLS skip verify (актуальный способ)?	2523
Как настроить зеркало для доступа к публичным registries (актуальный способ)?	2524
Как использовать NodeGroup с приоритетом?	2525
Как интерпретировать состояние группы узлов?	2526
Как заставить werf игнорировать состояние Ready в группе узлов?	2528
Что такое ресурс Instance?	2528
Когда требуется перезагрузка узлов?	2528
Как включить задержку выключения или перезагрузки узла, пока на нём работают критичные поды?	2529
Поиск подов, блокирующих выключение, и диагностика на узле	2529
Как механизм fencing обрабатывает разные типы узлов?	2529
Облачные узлы (CloudEphemeral, CloudPermanent)	2530
Статические узлы (Static, CloudStatic)	2530
Отключение автоматической перезагрузки после kernel panic	2530
Как работать с GPU-узлами?	2530
Порядок действий по добавлению GPU-узла в кластер	2530
Как выполнять мониторинг GPU?	2535
Какие режимы работы GPU поддерживаются?	2535
Как посмотреть доступные MIG-профили в кластере?	2535
Как задать свой MIG-профиль для каждой карты на узле?	2536
Для GPU не активируется MIG-профиль — что проверить?	2536
Поддерживаются ли AMD или Intel GPU?	2537
Примеры	2537
Примеры описания NodeGroup	2537
Облачные узлы	2537
Статические узлы	2537
Системные узлы	2537
Узлы с GPU	2538
Эксклюзивный режим (Exclusive)	2538
TimeSlicing (4 слота)	2539
MIG (профиль all-1g.5gb)	2539
Проверка работы: тестовая задача (CUDA vectoradd)	2539
Добавление статического узла в кластер	2540
Вручную	2540
С помощью Cluster API Provider Static	2540
С помощью Cluster API Provider Static для нескольких групп узлов	2542
Cluster API Provider Static: перемещение узлов между NodeGroup	2544
Исходная конфигурация	2544
Шаги по перемещению узла между NodeGroup	2545
1. Создание новой NodeGroup для целевой группы узлов	2545
2. Обновление лейбла у StaticInstance	2545
3. Уменьшение количества статических узлов в исходной NodeGroup	2546
Пример описания NodeUser	2546
Пример описания NodeGroupConfiguration	2546
Установка плагина cert-manager для kubectl на master-узлах	2546
Задание параметра sysctl	2546
Добавление корневого сертификата в хост	2547
Добавление в containerd возможности скачивать образы из insecure container registry	2548
Модуль okmeter	2548
Настройки	2548

Параметры	2550
Примеры	2551
Пример	2551
Модуль openvpn	2551
Варианты публикации VPN-сервиса	2551
Доступные ресурсы кластера после подключения к VPN	2552
Аудит пользовательских соединений	2552
Настройки	2552
Конверсии	2553
Параметры	2554
Аутентификация	2558
FAQ	2558
Почему не работает автоматическая настройка DNS-сервера при подключении на macOS и Linux с помощью клиента OpenVPN	2558
Как отозвать, ротировать или удалить сертификат клиента	2559
Как ротировать сертификат сервера	2559
Как ротировать корневой сертификат (CA)	2559
Примеры	2560
Пример для кластеров bare metal	2560
Пример для AWS и Google Cloud	2560
Пример для публичного IP-адреса на внешнем балансировщике	2560
Модуль operator-prometheus	2561
Prometheus	2561
Функции Prometheus	2561
Настройка Prometheus	2561
Источники целей мониторинга	2562
Prometheus Operator	2564
Функции Prometheus Operator	2564
Состав пода Prometheus	2564
Обработка ServiceMonitor	2564
Обработка кастомных ресурсов с правилами	2565
Настройки	2566
Параметры	2567
Custom resources	2569
PodMonitor	2569
Probe	2575
ScrapeConfig	2582
ServiceMonitor	2594
Prometheus-мониторинг	2601
Мониторинг аппаратных ресурсов	2601
Мониторинг Kubernetes	2601
Мониторинг Ingress	2601
Режим расширенного мониторинга	2601
Алертинг в режиме расширенного мониторинга	2601
Управление мониторингом как кодом (IaC подход)	2601
Алерты	2601
Отправка алертов во внешние системы	2602
Архитектура	2602
Базовые компоненты мониторинга	2602
Режим отказоустойчивости и высокой доступности мониторинга (HA)	2602
Настройки	2603
Требования	2604
Конверсии	2604
Параметры	2604
Аутентификация	2609
Примечание	2609
Custom resources	2609
ClusterAlert	2609
CustomAlertmanager	2610
CustomPrometheusRules	2640
GrafanaAdditionalDatasource	2641
GrafanaAlertsChannel	2643

GrafanaDashboardDefinition	2644
PrometheusRemoteWrite	2644
Примеры конфигурации	2647
Пример конфигурации модуля	2647
Запись данных Prometheus в longterm storage	2647
Пример минимального PrometheusRemoteWrite	2647
Пример расширенного PrometheusRemoteWrite	2647
Подключение Prometheus к сторонней Grafana	2648
Подключение стороннего приложения к Prometheus	2649
Отправка алертов в Telegram	2650
Пример отправки алертов в Slack с фильтром	2651
Пример отправки алертов в Opsgenie	2651
Пример отправки алертов по электронной почте	2652
FAQ	2653
Как собирать метрики с приложений, расположенных вне кластера?	2653
Пример	2653
Как добавить дополнительные дашборды в вашем проекте?	2653
Как добавить алерты и/или recording-правила для вашего проекта?	2654
Как подключить дополнительные data source для Grafana?	2655
Как обеспечить безопасный доступ к метрикам?	2655
Пример безопасного сбора метрик с приложения, расположенного в кластере	2655
Пример безопасного сбора метрик с приложения, расположенного вне кластера	2658
Как добавить Alertmanager?	2660
Как добавить внешний дополнительный Alertmanager?	2660
Как в Alertmanager игнорировать лишние алерты?	2661
Почему нельзя установить разный scrapeInterval для отдельных таргетов?	2662
Как ограничить потребление ресурсов Prometheus?	2662
Как настроить ServiceMonitor или PodMonitor для работы с Prometheus?	2662
Как настроить Probe для работы с Prometheus?	2663
Как настроить PrometheusRules для работы с Prometheus?	2663
Как увеличить размер диска	2664
Как получить информацию об алертах в кластере?	2664
Как добавить дополнительные эндпоинты в scrape config?	2665
Модуль prometheus-metrics-adapter	2666
Как работает	2666
Настройки	2666
Требования	2668
Параметры	2668
Custom resources	2668
Namespaced custom resources	2668
ServiceMetric	2668
IngressMetric	2669
PodMetric	2669
DeploymentMetric	2669
StatefulSetMetric	2669
NamespaceMetric	2669
DaemonSetMetric (недоступен пользователям)	2670
Cluster custom resources	2670
ClusterServiceMetric (недоступен пользователям)	2670
ClusterIngressMetric (недоступен пользователям)	2670
ClusterPodMetric (недоступен пользователям)	2670
ClusterDeploymentMetric (недоступен пользователям)	2670
ClusterStatefulSetMetric (недоступен пользователям)	2670
ClusterDaemonSetMetric (недоступен пользователям)	2671
Примеры конфигурации	2671
Типы масштабирования	2671
Классическое масштабирование по потреблению ресурсов	2672
Масштабирование по кастомным метрикам	2673
Регистрация кастомных метрик в Kubernetes API	2673
Применяем кастомные метрики в HPA	2673
Пример использования кастомных метрик с размером очереди RabbitMQ	2674
Пример использования нестабильной кастомной метрики	2675

Примеры с использованием кастомных метрик типа Pods	2676
Регистрация внешних метрик в Kubernetes API	2678
Применение внешних метрик в HPA	2679
Пример с размером очереди в Amazon SQS	2679
Способы отладки	2680
Как получить список кастомных метрик?	2680
Как получить значение метрики, привязанной к объекту?	2680
Как получить значение метрики, созданной через NamespaceMetric?	2681
Как получить external-метрики?	2681
Модуль Prometheus Pushgateway	2681
Настройки	2681
Требования	2683
Параметры	2683
Примеры	2683
Пример настройки модуля	2683
Отправка метрики	2684
Удаление метрик	2684
Модуль registry	2684
Описание	2684
Ограничения и особенности использования модуля	2685
Ограничения при установке кластера	2685
Ограничения по условиям работы	2685
Ограничения по переключению режимов	2685
Архитектура режима Direct	2685
Архитектура режима Proxy	2686
Архитектура режима Local	2687
Настройка	2687
Параметры	2689
FAQ	2691
Как мигрировать на модуль registry?	2691
Для containerd v2	2691
Для containerd v1	2692
Как мигрировать обратно с модуля registry?	2695
Как посмотреть статус переключения режима registry?	2698
Пример использования	2700
Переключение на режим Direct	2700
Переключение на режим Proxy	2702
Переключение на режим Local	2703
Переключение на режим Unmanaged	2706
Модуль registry-packages-proxy	2707
Архитектура	2708
HTTP API	2708
Иконки пакетов (/v1/packages/) — только изнутри кластера	2708
Загрузка Deckhouse CLI (/v1/images/) — публичный	2709
Внутренний эндпоинт /package	2710
Поток извлечения пакетов	2710
Высокая доступность	2710
Роли RBAC, создаваемые модулем	2710
Ограничения	2710
Настройки	2710
Модуль secret-copier	2712
Как работает?	2712
Что нужно настроить?	2713
Как ограничить список пространств имён, в которые будет производиться копирование?	2713
Модуль service-with-healthchecks	2713
Ограничения стандартного балансировщика Service	2713
Возможности балансировщика ServiceWithHealthcheck	2713
Внутреннее устройство балансировщика ServiceWithHealthcheck	2714
Настройки	2714
Параметры	2717
Custom Resources	2717

ServiceWithHealthchecks	2717
Примеры	2728
Размещение двух независимых балансировщиков на одной виртуальной машине	2729
Создание виртуальной машины	2729
Манифесты балансировщиков для веб-сервиса и SMTP	2729
Балансировщики для работы с PostgreSQL-кластером	2730
Создание StatefulSet для PostgreSQL	2730
Конфигурация балансировщиков ServiceWithHealthchecks	2731
Передача аннотаций и лейблов в дочерний Service	2732
Модуль terraform-manager	2732
Настройки	2733
Параметры	2734
Мониторинг SLA кластера	2735
Интерфейс	2735
Настройки	2736
Конверсии	2738
Параметры	2738
Аутентификация	2745
Кастомные ресурсы	2745
Downtime	2745
UptimeHookProbe	2746
UptimeRemoteWrite	2746
FAQ	2749
Почему периодически не могут разместиться поды uptime-probe-controller-manager, почему некоторые поды постоянно удаляются?	2749
Примеры	2749
Пример конфигурации remote_write	2749
Модуль user-authn	2749
Возможности интеграции	2750
Базовая аутентификация в API Kubernetes	2750
Интеграция с приложениями	2750
Веб-интерфейс для генерации готовых kubeconfig-файлов	2750
Публикация API kubernetes через Ingress	2751
Расширения от Фланта	2751
Отказоустойчивый режим	2751
Настройки	2751
Конверсии	2753
Параметры	2753
Custom Resources	2760
DexAuthenticator	2760
DexClient	2776
DexProvider	2778
Group	2801
UserOperation	2801
User	2804
Примеры конфигурации	2808
Пример конфигурации модуля	2808
Примеры настройки провайдера	2808
Проверка подключения провайдера	2808
GitHub	2808
GitLab	2809
Atlassian Crowd	2809
Bitbucket Cloud	2810
OIDC (OpenID Connect)	2810
Keycloak	2810
Okta	2811
Blitz Identity Provider	2812
LDAP	2813
Настройка базовой аутентификации	2813
Kerberos (SPNEGO) SSO для LDAP	2814
SAML	2815
Настройка OAuth2-клиента в Dex для подключения приложения	2816

Предоставление приложению доступа к Kubernetes API	2817
Локальная аутентификация	2817
Создание пользователя	2817
Операции над локальным пользователем	2818
Административные операции	2818
Сброс пароля пользователем	2819
Добавление пользователя в группу	2819
Парольная политика	2820
Двухфакторная аутентификация (2FA)	2821
Выдача прав пользователю или группе	2821
FAQ	2821
Как защитить мое приложение?	2821
Настройка ограничений на основе CIDR	2823
Как работает аутентификация с помощью DexAuthenticator	2823
Как сгенерировать kubeconfig для доступа к Kubernetes API?	2823
Настройка kube-apiserver	2824
Как работает подключение к Kubernetes API с помощью сгенерированного kubeconfig	2824
Как сменить секрет OAuth2-клиента kubernetes?	2824
Как включить SSO по Kerberos (SPNEGO) для LDAP?	2825
Как настроить базовую аутентификацию для доступа к Kubernetes API через LDAP?	2825
Как Dex защищен от подбора логина и пароля?	2825
Модуль user-authz	2825
Экспериментальная ролевая модель	2826
Use-роли	2826
Manage-роли	2827
Подсистемы ролевой модели	2828
Текущая ролевая модель	2830
Список доступа для каждой роли модуля по умолчанию	2830
Настройки	2833
Параметры	2835
Custom Resources	2835
AuthorizationRule	2835
ClusterAuthorizationRule	2837
Примеры конфигурации	2844
Пример назначения прав администратору кластера	2844
Пример назначения прав сетевому администратору	2845
Пример назначения административных прав пользователю в рамках пространства имён	2846
Пример ClusterAuthorizationRule	2847
Пример выдачи прав на все неймспейсы	2848
Создание пользователя	2849
Создание ServiceAccount для сервера и предоставление ему доступа	2849
Создание пользователя с помощью клиентского сертификата	2851
Создание пользователя с помощью сертификата, выпущенного через OpenSSL	2852
Создание пользователя с помощью сертификата, выпущенного через Kubernetes API	2853
Предоставление доступа созданному пользователю	2855
Настройка kube-apiserver для работы в режиме multi-tenancy	2855
Как проверить, что у пользователя есть доступ?	2856
Настройка прав высокоуровневых ролей	2857
FAQ	2858
Как создать пользователя?	2858
Как ограничить права пользователю конкретными пространствами имён?	2858
Что, если два ClusterAuthorizationRules подходят для одного пользователя?	2858
Как расширить роли или создать новую?	2859
Создание новой роли подсистемы	2859
Расширение пользовательской роли	2861
Расширение существующих manage subsystem-ролей	2862
Расширение manage subsystem-ролей с добавлением нового пространства имён	2862
Расширение существующих use-ролей	2863
Модуль vertical-pod-autoscaler	2863
Grafana dashboard	2864

Архитектура Vertical Pod Autoscaler	2864
Настройки	2864
Требования	2866
Параметры	2866
Custom resources	2867
VerticalPodAutoscaler	2867
VerticalPodAutoscalerCheckpoint	2878
FAQ	2881
Как посмотреть рекомендации Vertical Pod Autoscaler?	2881
Как Vertical Pod Autoscaler работает с лимитами?	2882
Пример 1	2882
Пример 2	2882
Примеры	2883
Настройка модуля	2883
Пример минимального custom resource VerticalPodAutoscaler	2883
Пример полного custom resource VerticalPodAutoscaler	2884

Документация Deckhouse Kubernetes Platform

Скачать в PDF

Приветствуем вас на главной странице документации Deckhouse Kubernetes Platform — платформы для управления Kubernetes-кластерами. Если вы еще не использовали платформу, рекомендуем начать с раздела Быстрый старт, где вы найдете пошаговые инструкции по развёртыванию платформы на любой инфраструктуре.

Как быстро найти то, что нужно:

- Если знаете, что вам необходимо — используйте поиск.
- Если нужен конкретный модуль — найдите его в списке.
- Для поиска по области применения воспользуйтесь меню.

Документация по Deckhouse Kubernetes Platform разных версий может отличаться. Выберите нужную версию в выпадающем списке вверху страницы. В списке доступны актуальные версии документации.

Если возникнут вопросы, вы можете обратиться за помощью в наш Telegram-канал. Мы обязательно поможем и проконсультируем.

Если вы используете коммерческую редакцию, можете написать нам на почту, мы также окажем вам поддержку.

Хотите улучшить Deckhouse Kubernetes Platform? Можете завести задачу, предложить свою идею или решение на GitHub.

А если вам хочется большего, присоединяйтесь к нашей команде! Мы рады новым специалистам.

Установка

i В разделе Быстрый старт доступны пошаговые инструкции по установке Deckhouse Kubernetes Platform. Попробуйте также графический установщик Deckhouse Kubernetes Platform.

На этой странице представлена обзорная информация по установке Deckhouse Kubernetes Platform (DKP).

Способы установки

Установить DKP можно следующими способами:

- с помощью CLI-установщика (доступен в виде образа контейнера и основан на утилите dhctl);
- с помощью графического установщика.

Далее рассмотрен процесс установки с помощью **CLI-установщика**.

Варианты установки

Установить DKP можно в следующих вариантах:

- **В поддерживаемом облаке.** Установщик автоматически создает и настраивает все необходимые ресурсы (включая виртуальные машины, сетевые объекты и т.д.), разворачивает кластер Kubernetes и устанавливает DKP. Полный список поддерживаемых облачных провайдеров доступен в разделе «Интеграция с IaaS».
- **На серверах bare metal (в том числе гибридные кластеры) или в неподдерживаемых облаках** Установщик настраивает указанные в конфигурации серверы или виртуальные машины, разворачивает кластер Kubernetes и устанавливает DKP. Пошаговые инструкции по развёртыванию на bare metal можно найти в разделе «Быстрый старт» → «Deckhouse Kubernetes Platform на bare metal».
- **В существующем кластере Kubernetes.** Установщик разворачивает DKP и интегрирует его с текущей инфраструктурой. Пошаговые инструкции по развёртыванию в существующем кластере можно найти в разделе «Быстрый старт» → «Deckhouse Kubernetes Platform в существующем кластере».

Требования к установке

Для оценки ресурсов, необходимых для установки Deckhouse Kubernetes Platform, вы можете ознакомиться со следующими руководствами:

- Руководство по подбору ресурсов для кластера на bare metal
- Руководство по разметке и объему дисков
- Руководство по подготовке к production

Перед установкой убедитесь в следующем:

- Для кластера на bare metal (в том числе гибридного кластера) и при установке в неподдерживаемых облаках: сервер использует операционную систему из списка поддерживаемых ОС или совместимую с ним, а также доступен по SSH через ключ.
- При настройке интеграции с поддерживаемыми облаками: имеются необходимые квоты для создания ресурсов и подготовлены параметры доступа к облачной инфраструктуре (зависят от конкретного провайдера).
- При наличии ограничений сетевого взаимодействия на уровне инфраструктуры соблюдены требования, описанные в разделе «Сетевое взаимодействие компонентов платформы».
- Есть доступ к хранилищу образов контейнеров Deckhouse (к публичному — `registry.deckhouse.io` или `registry.deckhouse.ru`, либо к зеркалу).

Подготовка конфигурации

Перед началом установки необходимо подготовить файл конфигурации установки, а также, при необходимости, `post-bootstrap`-скрипт.

Файл конфигурации установки

Файл конфигурации установки состоит из YAML-секций (документов) и содержит настройки DKP, а также описание (манифесты) объектов и ресурсов кластера, которые будут созданы после установки. Файл конфигурации установки используется в CLI-установщике и передается с помощью параметра `--config` (см. далее).

Список обязательных и опциональных объектов и ресурсов кластера, которые могут понадобиться в файле конфигурации установки:

1. `InitConfiguration` (**обязательный**) — начальные параметры конфигурации, необходимые для запуска DKP.

i Начиная с версии DKP 1.75, используйте `ModuleConfig deckhouse` для настройки доступа к хранилищу образов DKP. Настройка доступа с помощью `InitConfiguration` (параметры `imagesRepo`, `registryDockerCfg`, `registryScheme`, `registryCA`) считается устаревшим способом.

2. `ClusterConfiguration` — общие параметры кластера, такие как версия Kubernetes (компонентов control plane кластера), сетевые настройки, параметры CRI и т. д. Является **обязательным**, кроме случая, когда DKP устанавливается в уже существующий кластер Kubernetes.

3. `StaticClusterConfiguration` — параметры кластера, развертываемого на серверах `bare-metal` (в том числе гибридного кластера) или виртуальных машинах в неподдерживаемых облаках. Является **обязательным**, кроме случая, когда DKP устанавливается в уже существующий кластер Kubernetes.

Для добавления группы узлов (объект `NodeGroup`) под рабочую нагрузку в кластер могут понадобиться также объекты `StaticInstance` и `SSHCredentials`.

4. `<PROVIDER>ClusterConfiguration` — параметры интеграции с облачным провайдером. Является **обязательным** при интеграции DKP с поддерживаемой облачной инфраструктурой.

Примеры ресурсов, настраивающих интеграцию DKP с облачным провайдером:

- `AWSClusterConfiguration` — Amazon Web Services;
- `AzureClusterConfiguration` — Microsoft Azure;
- `DynamixClusterConfiguration` — Basis Dynamix;
- `DVPClusterConfiguration` — Deckhouse Virtualization Platform;
- `GCPClusterConfiguration` — Google Cloud Platform;
- `HuaweiCloudClusterConfiguration` — Huawei Cloud;
- `OpenStackClusterConfiguration` — OpenStack, OVHcloud, Selectel, VK Cloud;
- `VsphereClusterConfiguration` — VMware vSphere;
- `VCDClusterConfiguration` — VMware Cloud Director;
- `YandexClusterConfiguration` — Yandex Cloud;
- `ZvirtClusterConfiguration` — zVirt.

Для добавления облачных узлов в кластер также понадобятся объекты `<PROVIDER>InstanceClass` (например `YandexInstanceClass` для Yandex Cloud), которые описывают конфигурацию виртуальных машин в группе узлов (объект `NodeGroup`).

5. Конфигурации модулей DKP.

Каждый модуль настраивается (а также может быть включен или отключен) с помощью собственного объекта `ModuleConfig` с именем модуля (например, `ModuleConfig user-authn` для модуля `user-authn`). Допустимые параметры, которые можно указывать в объекте `ModuleConfig`, можно найти в документации соответствующего модуля в разделе «Настройки» (например, настройки модуля `user-authn`).

Список всех модулей Deckhouse Kubernetes Platform доступен в разделе «Модули» документации.

Некоторые модули могут быть включены и предварительно настроены автоматически, в зависимости от выбранного варианта установки и конфигурации кластера (например, модули, обеспечивающие работу control plane кластера и сети).

Модули, часто настраиваемые при установке:

- `global` — глобальные настройки DKP для указания параметров, которые используются по умолчанию всеми модулями и компонентами (шаблон DNS-имен, `StorageClass`, настройки расположения компонентов модулей и т.д.);
- `deckhouse` — настройки доступа к хранилищу образов, желаемый канал обновлений и другие параметры;
- `user-authn` — отвечает за единую систему аутентификации;
- `cni-cilium` — отвечает за работу сети в кластере (например, используется при установке DKP на `bare metal`, в закрытом окружении, на РЕД-виртуализации и на SpaceVM).

Если кластер изначально создается с узлами, выделенными для определенных типов нагрузки (например, системные узлы или узлы для мониторинга), рекомендуется в конфигурации модулей, использующих тома постоянного хранилища, явно задавать параметр `nodeSelector` (например, в параметре `nodeSelector ModuleConfig prometheus` для модуля `prometheus`).

6. `IngressNginxController` — параметры создаваемого балансировщика HTTP/HTTPS-трафика (Ingress-контроллера).
7. `NodeGroup` — параметры группы узлов. Необходим для добавления узлов под рабочую нагрузку в кластер.
8. Объекты для настройки аутентификации и авторизации, такие как `ClusterAuthorizationRule`, `AuthorizationRule`,

User, Group, DexProvider.

Читайте подробнее в документации о настройке аутентификации и авторизации.

Пример файла конфигурации установки...

- Конфигурация, применимая с версии 1.75 DKP
- Устаревший вариант конфигурации

В этом примере доступ к хранилищу образов DKP настраивается с помощью ModuleConfig `deckhouse`.

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
podSubnetCIDR: 10.111.0.0/16
serviceSubnetCIDR: 10.222.0.0/16
kubernetesVersion: "Automatic"
clusterDomain: cluster.local
---
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
vNetCIDR: 10.241.0.0/16
subnetCIDR: 10.241.0.0/24
masterNodeGroup:
  replicas: 3
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
    enableExternalIP: true
provider:
  subscriptionId: <SUBSCRIPTION_ID>
  clientId: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
  tenantId: <TENANT_ID>
  location: westeurope
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
  settings:
    releaseChannel: Stable
    bundle: Default
    logLevel: Info
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: test-registry.io/some/path
        scheme: HTTPS
        username: <username>
        password: <password>
        ca: <CA>
  version: 1
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  settings:
    modules:
      publicDomainTemplate: "%s.k8s.example.com"
  version: 2
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
```

```

spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: DoNotNeed
    publishAPI:
      enabled: true
      https:
        mode: Global
      global:
        kubeconfigGeneratorMasterCA: ""
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 1
  enabled: true
  settings:
    earlyOomEnabled: false
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Укажите в случае использования выделенных узлов для мониторинга.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
    nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
    - kind: User

```

```

  name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1Xa0Gg2'
```

В этом примере доступ к хранилищу образов DKP настраивается с помощью InitConfiguration.

```

apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
podSubnetCIDR: 10.111.0.0/16
serviceSubnetCIDR: 10.222.0.0/16
kubernetesVersion: "Automatic"
clusterDomain: cluster.local
---
apiVersion: deckhouse.io/v1
kind: InitConfiguration
deckhouse:
  imagesRepo: registry.deckhouse.ru/deckhouse/ee
  registryDockerCfg: eyJhdXRocyI6IHsgInJlZ2zzzmRlY2tob3Vxcxcxc5ydSI6Isssfx0K
---
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
vNetCIDR: 10.241.0.0/16
subnetCIDR: 10.241.0.0/24
masterNodeGroup:
  replicas: 3
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
    enableExternalIP: true
provider:
  subscriptionId: <SUBSCRIPTION_ID>
  clientId: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
  tenantId: <TENANT_ID>
  location: westeurope
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
  settings:
    releaseChannel: Stable
    bundle: Default
    logLevel: Info
  version: 1
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  settings:
    modules:
      publicDomainTemplate: "%s.k8s.example.com"
  version: 2
---
apiVersion: deckhouse.io/v1alpha1
```

```

kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: DoNotNeed
    publishAPI:
      enabled: true
      https:
        mode: Global
        global:
          kubeconfigGeneratorMasterCA: ""
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 1
  enabled: true
  settings:
    earlyOomEnabled: false
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Укажите в случае использования выделенных узлов для мониторинга.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
    nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin

```

```

spec:
  subjects:
  - kind: User
    name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1Xa0Gg2'

```

Post-bootstrap-скрипт

Установщик позволяет выполнить пользовательский скрипт на одном из master-узлов после завершения установки (post-bootstrap-скрипт). Такой скрипт может использоваться для:

- дополнительной настройки кластера;
- сбора диагностической информации;
- интеграции с внешними системами и других задач.

Указать путь к post-bootstrap-скрипту можно с помощью параметра `--post-bootstrap-script-path` при запуске CLI-установщика.

Пример скрипта, выводящего IP-адрес балансировщика...

Пример скрипта, который выводит IP-адрес балансировщика после установки DKP:

```

#!/usr/bin/env bash

set -e
set -o pipefail

INGRESS_NAME="nginx"

echo_err() { echo "$@" 1>&2; }

# Объявление переменной.
lb_ip=""

# Получение IP-адреса балансировщика нагрузки.
for i in {0..100}
do
  if lb_ip="$(kubectl -n d8-ingress-nginx get svc "${INGRESS_NAME}-load-balancer" -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'); then
    if [ -n "$lb_ip" ]; then
      break
    fi
  fi

  lb_ip=""

  sleep 5
done

if [ -n "$lb_ip" ]; then
  echo_err "The load balancer external IP: $lb_ip"
else
  echo_err "Could not get the external IP of the load balancer"
  exit 1
fi

outContent="{\"frontend_ips\": [\"$lb_ip\"]}"

if [ -z "$OUTPUT" ]; then
  echo_err "The OUTPUT env is empty. The result was not saved to the output file."
else
  echo "$outContent" > "$OUTPUT"
fi

```

Установка

i При установке коммерческой редакции Deckhouse Kubernetes Platform из публичного хранилища образов `registry.deckhouse.ru` необходимо предварительно пройти аутентификацию с использованием лицензионного ключа:

```
docker login -u license-token registry.deckhouse.ru
```

Команда для запуска контейнера с установщиком из публичного хранилища образов контейнеров Deckhouse:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] registry.deckhouse.ru/deckhouse/<DECKHOUSE_REVISION>/install:
<RELEASE_CHANNEL> bash
```

Где:

1. `<DECKHOUSE_REVISION>` — редакция DKP. Например, `ee` — для Enterprise Edition, `ce` — для Community Edition и т. д.
2. `<MOUNT_OPTIONS>` — параметры монтирования файлов в контейнер установщика, таких как:
 - SSH-ключи доступа;
 - файл конфигурации;
 - файл ресурсов и т. д.
3. `<RELEASE_CHANNEL>` — канал обновлений в формате kebab-case:
 - `alpha` — для канала обновлений Alpha;
 - `beta` — для канала обновлений Beta;
 - `early-access` — для канала обновлений Early Access;
 - `stable` — для канала обновлений Stable;
 - `rock-solid` — для канала обновлений Rock Solid.

Пример команды для запуска контейнера с установщиком DKP Community Edition из канала обновлений Stable:

```
docker run -it --pull=always \
-v "$PWD/config.yaml:/config.yaml" \
-v "$PWD/dhctl-tmp:/tmp/dhctl" \
-v "$HOME/.ssh:/tmp/.ssh/" registry.deckhouse.ru/deckhouse/ce/install:stable bash
```

Установка DKP осуществляется в контейнере установщика с помощью команды `dhctl`:

- Для запуска установки DKP с развертыванием нового кластера (все случаи, кроме установки в существующий кластер) используйте команду `dhctl bootstrap`.
- Для установки DKP в уже существующий кластер используйте команду `dhctl bootstrap-phase install-deckhouse`.

i Для получения подробной справки по параметрам команды выполните `dhctl bootstrap -h`.

Пример запуска установки DKP с развертыванием кластера в облаке:

```
dhctl bootstrap \
--ssh-user=<SSH_USER> --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
--config=/config.yaml
```

Где:

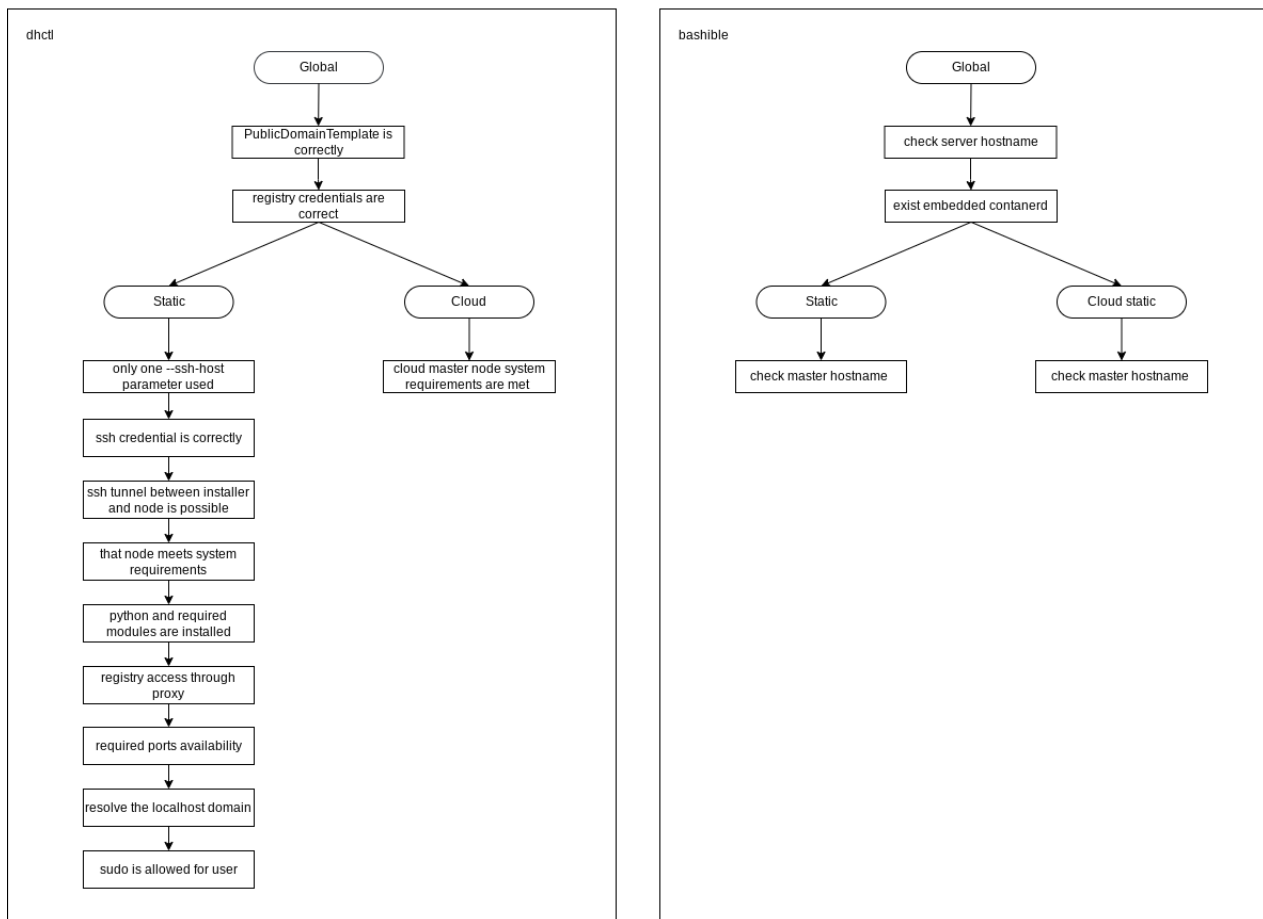
- `/config.yaml` — файл конфигурации установки;
- `<SSH_USER>` — имя пользователя для подключения по SSH к серверу;
- `--ssh-agent-private-keys` — файл приватного SSH-ключа для подключения по SSH.
- `<SSH_PRIVATE_KEY_FILE>` — имя приватного ключа. Например, для ключа с RSA-шифрованием это может быть `id_rsa`, а для ключа с ED25519-шифрованием — `id_ed25519`.

Проверки перед началом установки

i Начиная с версии 1.74, модули DKP устанавливаются в виде образов в формате EROFS, подключаемых только для чтения, что защищает их от изменения после установки. Этот механизм включается автоматически, если на узле, где работает контроллер DKP (по умолчанию — master-узел), в ядре зарегистрирована файловая система `erofs`. DKP загружает этот модуль ядра только на узлах с `containerd v2`, поэтому при использовании `containerd v1` на master-узлах загрузку `erofs` нужно обеспечить средствами операционной системы. Иначе DKP будет устанавливать модули обычным способом, без защиты их целостности и без отдельного алерта. Подробнее — в разделе «Защита целостности модулей DKP».

Схема выполнения проверок, выполняемых установщиком перед началом установки...

Preflight checks



Список проверок, выполняемых установщиком перед началом установки Deckhouse Kubernetes Platform:

1. Общие проверки:

- Значения параметров `publicDomainTemplate` и `clusterDomain` не совпадают.
- Данные аутентификации для хранилища образов, указанные в конфигурации установки, корректны.
- Имя хоста соответствует следующим требованиям:
 - длина не более 63 символов;
 - состоит только из строчных букв;
 - не содержит спецсимволов (допускаются символы `-` (дефис) и `.` (точка), при этом они не могут быть в начале или в конце имени).
- На сервере (ВМ) установлен поддерживаемый container runtime (`containerd`).
- Имя хоста уникально в пределах кластера.
- На сервере установлено корректное время.
- Адресное пространство подов (`podSubnetCIDR`) и сервисов (`serviceSubnetCIDR`) кластера не пересекаются.

2. Проверки для установки статического и гибридного кластера:

- Указан только один параметр `--ssh-host`. Для статической конфигурации кластера можно задать только один IP-адрес для настройки первого master-узла.
- Должна быть возможность подключения по SSH с использованием указанных данных аутентификации.
- Должна быть возможность установки SSH-туннеля до сервера (или виртуальной машины) master-узла.
- Сервер (ВМ), выбранный для установки master-узла, должен соответствовать минимальным системным требованиям:
 - не менее 4 CPU;
 - не менее 8 ГБ RAM;
 - не менее 60 ГБ диска с производительностью 400+ IOPS;
 - ядро Linux версии 5.8 или новее;
 - установлен один из пакетных менеджеров: `apt`, `apt-get`, `yum` или `rpm`;
 - доступ к стандартным системным репозиториям для установки зависимостей;
 - в случае с РЕД ОС — убедитесь, что установлены `yum` и `which` (по умолчанию могут отсутствовать);
 - в случае использования ContainerdV2** в качестве container runtime по умолчанию на узлах кластера:
 - поддержка `CgroupsV2`;
 - `systemd` версии 244;
 - поддержка модуля ядра `erofs`.
- На сервере (ВМ) для master-узла должен быть доступен Python (Python 3 или Python 2). Также должны быть доступны стандартные модули Python, используемые установщиком:
 - для работы с HTTP-запросами: `urllib.request` (Python 3) или `urllib2` (Python 2);
 - для обработки HTTP-ошибок: `urllib.error` (Python 3) или `urllib2` (Python 2);
 - для работы с конфигурацией: `configparser` (Python 3) или `ConfigParser` (Python 2);
 - для запуска HTTP-сервера: `http.server` (Python 3) или `SimpleHTTPServer` (Python 2);
 - для работы с TCP-сервером: `http.server` (Python 3) или `SocketServer` (Python 2).
- Хранилище образов доступно через прокси (если настройки прокси указаны в конфигурации установки).
- На сервере (ВМ) для master-узла и на хосте, с которого запускается установщик, должен быть доступен

порт 22/TCP (сетевой доступ от персонального компьютера), необходимый для процесса установки.

Подробнее в разделе «Сетевое взаимодействие компонентов платформы».

- DNS должен разрешать localhost в IP-адрес 127.0.0.1.
- На сервере (БМ) пользователю доступна команда `sudo`.
- Открыты необходимые порты для установки:
 - между хостом запуска установщика и сервером — порт 22/TCP;
 - отсутствуют конфликты по портам, которые используются процессом установки.
- На сервере (БМ) установлено корректное время.
- Адресное пространство подов (`podSubnetCIDR`), сервисов (`serviceSubnetCIDR`) и внутренней сети кластера (`internalNetworkCIDRs`) не пересекаются.
- На сервере (БМ) отсутствует пользователь `deckhouse`.

3. Проверки для установки облачного кластера:

- Конфигурация виртуальной машины master-узла удовлетворяет минимальным требованиям.
- API облачного провайдера доступно с узлов кластера.
- Проверка конфигурации Yandex Cloud с NAT Instance.

Список флагов пропуска проверок...

Для пропуска отдельной проверки используйте флаг `--preflight-skip-check`, передав в качестве аргумента имя preflight-чека. Флаг можно указывать несколько раз.

- `--preflight-skip-all-checks` — пропуск всех предварительных проверок;
- `--preflight-skip-check=static-ssh-tunnel` — пропуск проверки проброса SSH;
- `--preflight-skip-check=ports-availability` — пропуск проверки доступности необходимых портов;
- `--preflight-skip-check=resolve-localhost` — пропуск проверки разрешения `localhost`;
- `--preflight-skip-check=dhctl-edition` — пропуск проверки версии DKP;
- `--preflight-skip-check=registry-access-through-proxy` — пропуск проверки доступа к хранилищу образов через прокси-сервер;
- `--preflight-skip-check=public-domain-template` — пропуск проверки шаблона `publicDomain`;
- `--preflight-skip-check=static-ssh-credential` — пропуск проверки учетных данных SSH-пользователя;
- `--preflight-skip-check=registry-credentials` — пропуск проверки учетных данных для доступа к хранилищу образов;
- `--preflight-skip-check=python-modules` — пропуск проверки наличия Python;
- `--preflight-skip-check=sudo-allowed` — пропуск проверки прав доступа для выполнения команды `sudo`;
- `--preflight-skip-check=static-system-requirements` — пропуск проверки соответствия системным требованиям;
- `--preflight-skip-check=static-single-ssh-host` — пропуск проверки количества указанных SSH-хостов;
- `--preflight-skip-check=cloud-api-accessibility` — пропуск проверки доступности Cloud API;
- `--preflight-skip-check=time-drift` — пропуск проверки отсутствия рассинхронизации времени (time drift);
- `--preflight-skip-check=cidr-intersection` — пропуск проверки пересечения CIDR;
- `--preflight-skip-check=deckhouse-user` — пропуск проверки наличия пользователя `deckhouse`;
- `--preflight-skip-check=yandex-cloud-config` — пропуск проверки конфигурации Yandex Cloud с `WithNatInstance`;
- `--preflight-skip-check=dvp-kubeconfig` — пропуск проверки DVP kubeconfig.
- `--preflight-skip-check=static-instances-ssh-credentials` — пропуск проверки доступности `StaticInstances` с `SSHCredentials`.

Пример применения флага пропуска:

```
dhctl bootstrap \
  --ssh-user=<SSH_USER> --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
  --config=/config.yml \
  --preflight-skip-all-checks
```



Замените здесь `<SSH_PRIVATE_KEY_FILE>` на имя вашего приватного ключа. Например, для ключа с RSA-шифрованием это может быть `id_rsa`, а для ключа с ED25519-шифрованием — `id_ed25519`.

Откат установки

Если установка была прервана или возникли проблемы во время установки в поддерживаемом облаке, то могут остаться ресурсы, созданные в процессе установки. Для их удаления выполните следующую команду в контейнере с установщиком:

```
dhctl bootstrap-phase abort
```



Файл конфигурации, передаваемый через параметр `--config` при запуске установщика, должен быть тем же, который использовался для первоначальной установки.

Закрытое окружение, работа через прокси-сервер и стороннее хранилище образов контейнеров



Подробнее с установкой и обновлением DKP в закрытом окружении можно ознакомиться в руководствах «Установка DKP в закрытом окружении» и «Обновление DKP в закрытом окружении».

Установка из стороннего хранилища образов контейнеров

DKP можно установить из стороннего хранилища образов или через проксирующий сервер внутри закрытого контура.



DKP поддерживает аутентификацию в хранилище образов по схемам Basic и Bearer token (сначала проверяется Basic, при неуспехе — Bearer).

Если перед хранилищем стоит прокси-сервер, он должен корректно проксировать заголовок Registry API v2 `Docker-Distribution-API-Version: registry/2.0`, иначе проверка Basic может завершиться ошибкой, а последующая попытка Bearer сообщением с ошибкой `couldn't find bearer realm parameter`.

Протестирована и гарантируется работа со следующими хранилищами образов: Nexus, Harbor, Artifactory, CNCF Distribution, Quay.

При работе со сторонним хранилищем образов не используйте учетную запись администратора для доступа к нему со стороны DKP. Используйте отдельную учетную запись с правами только на чтение и только в пределах нужного раздела в хранилище образов. Ознакомьтесь с примером создания такой учетной записи.

Варианты настройки работы со сторонними хранилищами образов при установке кластера:

- начиная с версии DKP 1.75 — с помощью ModuleConfig `deckhouse`;
- до версии DKP 1.75 — с помощью InitConfiguration (устаревший способ, пример приведен ниже).

Для настройки с помощью ModuleConfig `deckhouse` укажите параметры доступа к стороннему хранилищу образов в секции `settings.registry`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        imagesRepo: test-registry.io/some/path
        scheme: HTTPS
        username: <username>
        password: <password>
        ca: <CA>
```

Настройка работы со сторонним хранилищем образов через InitConfiguration ****(устаревший способ)****

Установите следующие параметры в InitConfiguration:

- `imagesRepo`: `<PROXY_REGISTRY>/<DECKHOUSE_REPO_PATH>/ee` — адрес образа DKP EE в стороннем хранилище образов. Пример: `imagesRepo: registry.deckhouse.ru/deckhouse/ee`;
- `registryDockerCfg`: `<BASE64>` — права доступа к стороннему хранилищу образов, зашифрованные в Base64.

Если разрешен анонимный доступ к образам DKP в стороннем хранилище образов, `registryDockerCfg` должен выглядеть следующим образом:

```
{"auths": { "<PROXY_REGISTRY>": {}}}
```

Приведенное значение должно быть закодировано в Base64.

Если для доступа к образам DKP в стороннем хранилище образов необходима аутентификация, `registryDockerCfg` должен выглядеть следующим образом:

```
{"auths": { "<PROXY_REGISTRY>": {"username": "<PROXY_USERNAME>", "password": "<PROXY_PASSWORD>", "auth": "<AUTH_BASE64>"}}}
```

где:

- `<PROXY_USERNAME>` — имя пользователя для аутентификации на `<PROXY_REGISTRY>`;
- `<PROXY_PASSWORD>` — пароль пользователя для аутентификации на `<PROXY_REGISTRY>`;
- `<PROXY_REGISTRY>` — адрес стороннего хранилища образов в виде `<HOSTNAME>[:PORT]`;
- `<AUTH_BASE64>` — строка вида `<PROXY_USERNAME>:<PROXY_PASSWORD>`, закодированная в Base64.

Итоговое значение для `registryDockerCfg` должно быть также закодировано в Base64.

Вы можете использовать следующий скрипт для генерации `registryDockerCfg`:

```

declare MYUSER='<PROXY_USERNAME>'
declare MYPASSWORD='<PROXY_PASSWORD>'
declare MYREGISTRY='<PROXY_REGISTRY>'

MYAUTH=$(echo -n "$MYUSER:$MYPASSWORD" | base64 -w0)
MYRESULTSTRING=$(echo -n "{\"auths\":{\"\"$MYREGISTRY\":{
{\"username\":\"$MYUSER\", \"password\":\"$MYPASSWORD\", \"auth\":\"$MYAUTH\"}}}" | base64 -w0)

echo "$MYRESULTSTRING"

```

Для настройки нестандартных конфигураций сторонних хранилищ образов в `InitConfiguration` предусмотрены еще два параметра:

- `registryCA` — корневой сертификат, которым можно проверить сертификат хранилища образов (если хранилище образов использует самоподписанные сертификаты);
- `registryScheme` — протокол доступа к хранилищу образов (`HTTP` или `HTTPS`). По умолчанию — `HTTPS`.

Особенности настройки Nexus

⚠ При взаимодействии с репозиторием типа `docker`, расположенным в Nexus (например, при выполнении команд `docker pull`, `docker push`), требуется указывать адрес в формате `<NEXUS_URL>:<REPOSITORY_PORT>/<PATH>`.

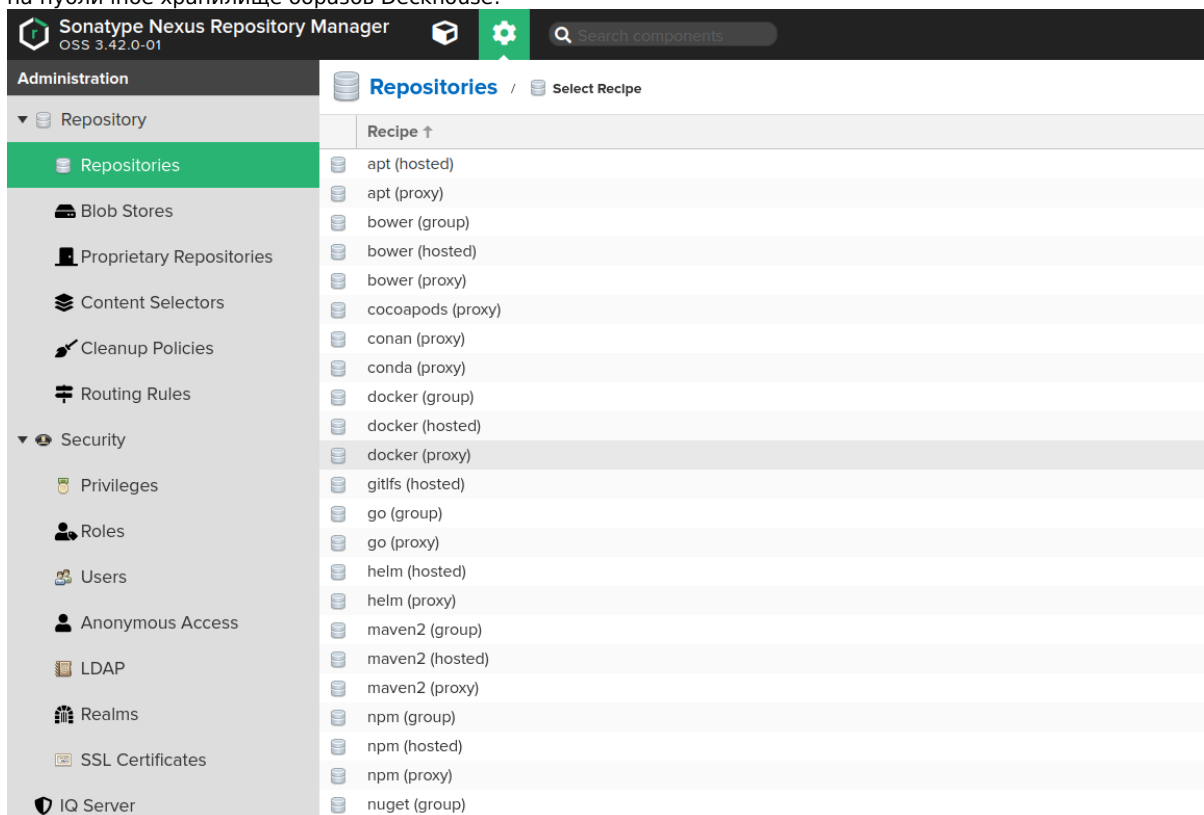
Использование значения `URL` из параметров репозитория Nexus **недопустимо**.

При использовании менеджера репозитория Nexus должны быть выполнены следующие требования:

- Создан **проксирующий** репозиторий Docker («Administration» → «Repository» → «Repositories»):
 - установлен в `0` параметр `Maximum metadata age` для репозитория.
- Настроен контроль доступа:
 - создана роль **Nexus** («Administration» → «Security» → «Roles») со следующими полномочиями:
 - `nx-repository-view-docker-<репозиторий>-browse`;
 - `nx-repository-view-docker-<репозиторий>-read`;
 - создан пользователь («Administration» → «Security» → «Users») с ролью **Nexus**.

Чтобы настроить Nexus, выполните следующие шаги:

1. Создайте проксирующий репозиторий Docker («Administration» → «Repository» → «Repositories»), указывающий на публичное хранилище образов Deckhouse.



2. Заполните поля страницы создания репозитория следующим образом:
 - Name должно содержать имя создаваемого репозитория, например, `d8-proxy`.
 - Repository Connectors / HTTP или Repository Connectors / HTTPS должно содержать выделенный порт для создаваемого репозитория, например, `8123` или иной.
 - Remote storage должно иметь значение `https://registry.deckhouse.ru/`.
 - Auto blocking enabled и Not found cache enabled могут быть выключены для отладки; в противном случае их следует включить.
 - Maximum Metadata Age должно быть равно `0`.
 - Если планируется использовать коммерческую редакцию Deckhouse Kubernetes Platform, флажок

Authentication должен быть включен, а связанные поля должны быть заполнены следующим образом:

- Authentication Type должно иметь значение Username.
- Username должно иметь значение license-token.
- Password должно содержать ключ лицензии Deckhouse Kubernetes Platform.

The screenshot displays the Sonatype Nexus Repository Manager OSS 3.43.0-01 interface. The left sidebar shows the 'Administration' menu with 'Repositories' selected. The main content area shows the configuration for the 'd8-proxy' repository. The 'Settings' tab is active, displaying the following information:

- Name:** d8-proxy
- Format:** docker
- Type:** proxy
- URL:** https://nexus.g-maiorov.hf.flant.com/repository/d8-proxy/
- Online:** ☒ If checked, the repository accepts incoming requests

Below the settings, the 'Repository Connectors' section provides information about Docker connectors and their usage. The 'HTTP' section shows a dropdown menu with '8123' selected. The 'HTTPS' section shows a dropdown menu with an empty selection. The 'Allow anonymous docker pull' section has a checkbox labeled 'Allow anonymous docker pull (Docker Bearer Token Realm required)'. The 'Docker Registry API Support' section has a checkbox labeled 'Enable Docker V1 API: Allow clients to use the V1 API to interact with this repository'. The 'Proxy' section has a 'Remote storage' field.

 Sonatype Nexus Repository Manager
OSS 3.43.0-01





Administration

▼  Repository

 Repositories

 Blob Stores

 Proprietary Repositories

 Content Selectors

 Cleanup Policies

 Routing Rules

▼  Security

 Privileges

 Roles

 Users

 Anonymous Access

 LDAP

 Realms

 SSL Certificates

 IQ Server

►  Support

►  System

 **Repositories** /  d8-proxy

 Delete repository

 Rebuild index

 Invalidate cache

 Enable HealthCheck

Settings

Proxy

Remote storage:
Location of the remote repository being proxied, e.g. `https://registry-1.docker.io`

Use the Nexus Repository truststore:
☐ Use certificates stored in the Nexus Repository truststore to connect to external systems  View certificates

Docker Index:
☒ Use proxy registry (specified above)
☐ Use Docker Hub
☐ Custom index

Foreign Layer Caching:
☐ Allow Nexus Repository Manager to download and cache foreign layers

Blocked:
☐ Block outbound connections on the repository

Auto blocking enabled:
☒ Auto-block outbound connections on the repository if remote peer is detected as unreachable/unresponsive

Maximum component age:
How long (in minutes) to cache artifacts before rechecking the remote repository. Release repositories should use -1.

Maximum metadata age:
How long (in minutes) to cache metadata before rechecking the remote repository.

Storage

Remote storage:

Sonatype Nexus Repository Manager
OSS 3.43.0-01

Administration

- Repository
- Repositories**
- Blob Stores
- Proprietary Repositories
- Content Selectors
- Cleanup Policies
- Routing Rules
- Security
 - Privileges
 - Roles
 - Users
 - Anonymous Access
 - LDAP
 - Realms
 - SSL Certificates
 - IQ Server
- Support
- System

Repositories / d8-proxy

Delete repository Rebuild index Invalidate cache Enable HealthCheck

Settings

Cleanup Policies:
Components that match any of the Applied policies will be deleted

Available Applied

Filter

HTTP

☒ Authentication

Authentication type:
Username

Username:
license-token

Password:
.....

☐ HTTP request settings

3. Настройте контроль доступа Nexus для доступа DKP к созданному репозиторию:

- Создайте роль **Nexus** («Administration» → «Security» → «Roles») с полномочиями `nx-repository-view-docker-<репозиторий>-browse` и `nx-repository-view-docker-<репозиторий>-read`.

Sonatype Nexus Repository Manager
OSS 3.43.0-01

Administration

- Repository
 - Repositories
 - Blob Stores
 - Proprietary Repositories
 - Content Selectors
 - Cleanup Policies
 - Routing Rules
- Security
 - Privileges
 - Roles**
 - Users
 - Anonymous Access
 - LDAP
 - Realms
 - SSL Certificates
 - IQ Server
 - Support
 - System

Type
Nexus role

Role Setup

Role ID
d8-access ✓

Role Name
d8-access ✓

Role Description *Optional*

Privileges

Available

- Filter
- Transfer All
- nx-repository-view-docker-*delete
- nx-repository-view-docker-*edit
- nx-repository-view-docker-*read

Given

- Filter
- Remove All
- nx-repository-view-docker-d8-prox...
- nx-repository-view-docker-d8-prox...

- Создайте пользователя («Administration» → «Security» → «Users») с ролью, созданной выше.

Sonatype Nexus Repository Manager
OSS 3.43.0-01

Administration

- Repository
 - Repositories
 - Blob Stores
 - Proprietary Repositories
 - Content Selectors
 - Cleanup Policies
 - Routing Rules
- Security
 - Privileges
 - Roles
 - Users**
 - Anonymous Access
 - LDAP
 - Realms
 - SSL Certificates
 - IQ Server
 - Support
 - System

Users / Create User

ID:
This will be used as the username
d8-access

First name:
d8-access

Last name:
d8-access

Email:
Used for notifications
d8-access@localhost

Password:
.....

Confirm password:
.....

Status:
Active

Roles:

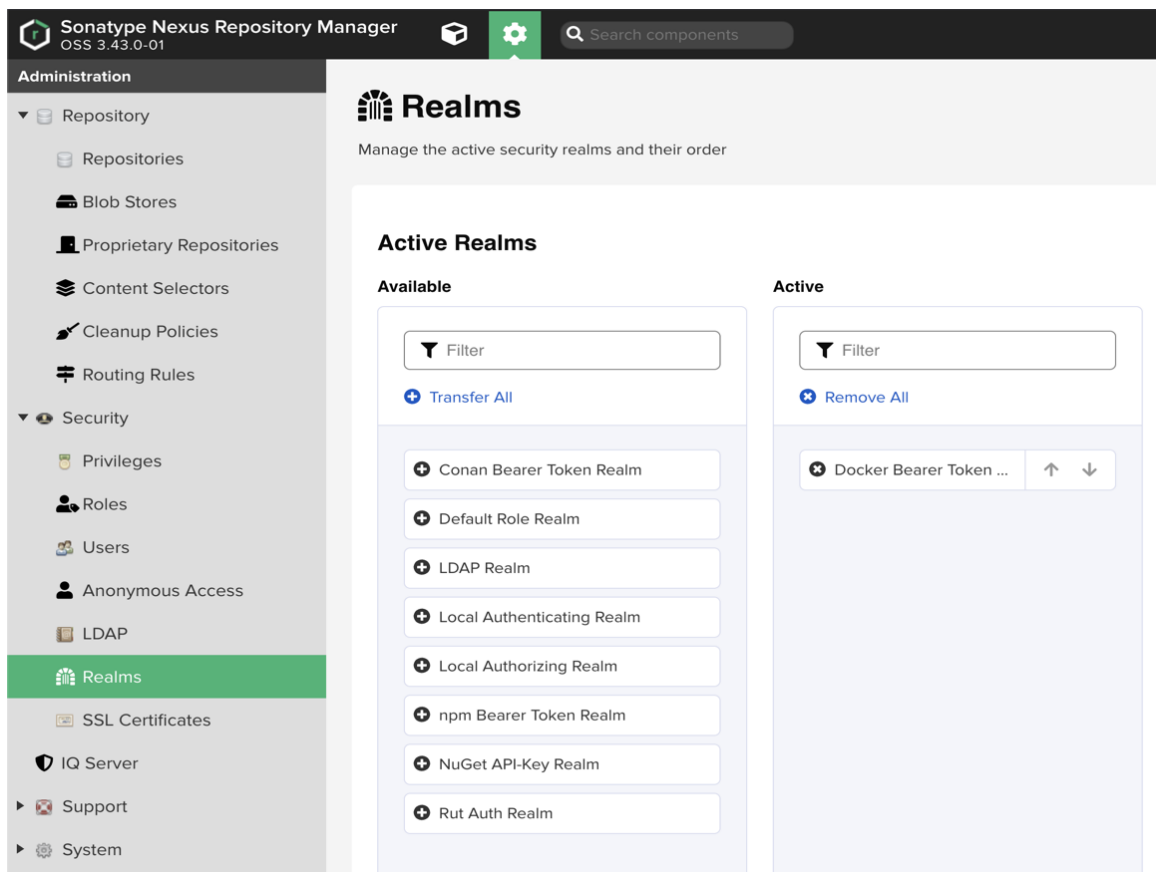
Available

- Filter
- nx-admin
- nx-anonymous

Granted

- d8-access

- Включите **Docker Bearer Token Realm** («Administration» → «Security» → «Realms»):
 - **Docker Bearer Token Realm** должен быть в списке **Active** (справа), а не в **Available** (слева).
 - Если его нет в **Active**:
 1. Найдите в списке **Available**.
 2. Переместите стрелкой в **Active**.
 3. Нажмите **Save**.
 4. **Перезапустите Nexus** (это обязательно, для применения изменений).

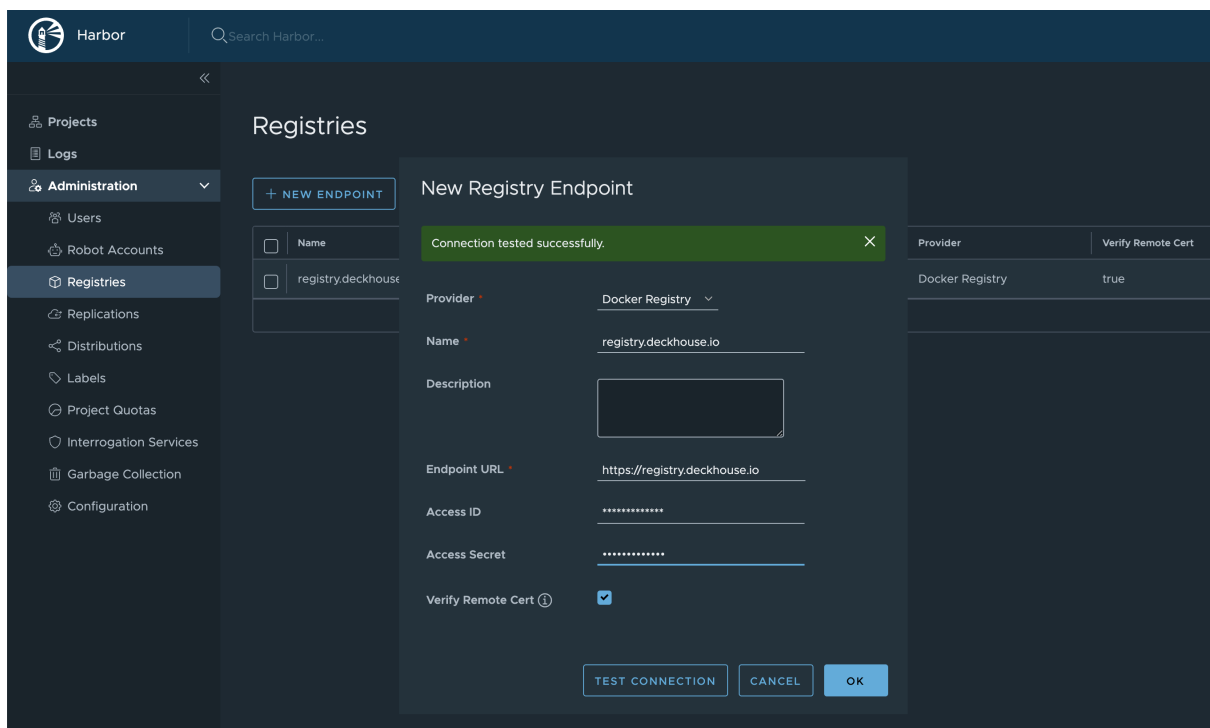


В результате образы DKP будут доступны, например, по следующему адресу: `https://<NEXUS_HOST>:<REPOSITORY_PORT>/deckhouse/ee:<d8s-version>`.

Особенности настройки Harbor

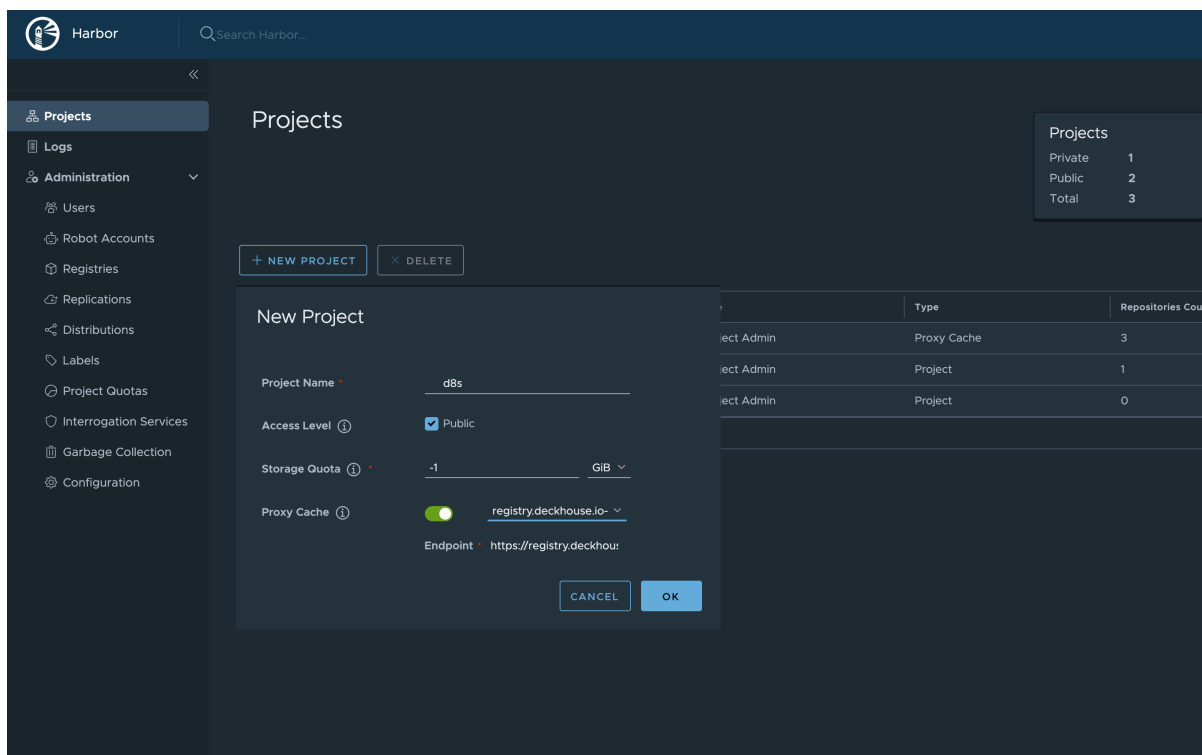
Используйте функцию Harbor Proxy Cache.

1. Настройте доступ к хранилищу образов:
 - в боковом меню перейдите в раздел «Administration» → «Registries» и нажмите «New Endpoint», чтобы добавить эндпоинт для хранилища образов;
 - в выпадающем списке «Provider» выберите «Docker Registry»;
 - в поле «Name» укажите имя эндпоинта на свое усмотрение;
 - в поле «Endpoint URL» укажите `https://registry.deckhouse.ru`;
 - в поле «Access ID» укажите `license-token`;
 - в поле «Access Secret» укажите свой лицензионный ключ Deckhouse Kubernetes Platform;
 - задайте остальные параметры по своему усмотрению;
 - нажмите «OK», чтобы подтвердить создание эндпоинта для хранилища образов.



2. Создайте новый проект:

- о в боковом меню перейдите в раздел «Projects» и нажмите «New Project», чтобы добавить проект;
- о в поле «Project Name» укажите любое имя проекта на свое усмотрение (например, `d8s`). Указанное имя будет частью URL-адреса;
- о в поле «Access Level» выберите «Public»;
- о включите «Proxy Cache» и в выпадающем списке выберите хранилище образов, созданное ранее;
- о задайте остальные параметры по своему усмотрению;
- о нажмите «OK», чтобы подтвердить создание проекта.



После настройки Harbor образы DKP станут доступны по адресу следующего вида: `https://your-harbor.com/d8s/deckhouse/ee:{d8s-version}`.

Ручная загрузка образов DKP и БД уязвимостей в приватное хранилище образов контейнеров

⚠ Утилита `d8 mirror` недоступна для использования с редакциями Community Edition (CE) и Basic Edition (BE).

i О текущем статусе версий на каналах обновлений можно узнать на releases.deckhouse.ru.

- Скачайте и установите утилиту Deckhouse CLI.
- Скачайте образы DKP в выделенную директорию, используя команду `d8 mirror pull`.

По умолчанию `d8 mirror pull` скачивает только актуальные версии DKP, базы данных сканера уязвимостей (если они входят в редакцию DKP) и официально поставляемых модулей. Например, для Deckhouse Kubernetes Platform 1.59 будет скачана только версия 1.59.12, т. к. этого достаточно для обновления платформы с 1.58 до 1.59.

Выполните следующую команду (укажите код редакции и лицензионный ключ), чтобы скачать образы актуальных версий:

```
d8 mirror pull \
  --source='registry.deckhouse.ru/deckhouse/<EDITION>' \
  --license='<LICENSE_KEY>' /home/user/d8-bundle
```

где:

- о `--source` — адрес хранилища образов Deckhouse;
- о `<EDITION>` — код редакции Deckhouse Kubernetes Platform (например, `ee`, `se`, `se-plus`). По умолчанию параметр `--source` ссылается на редакцию Enterprise Edition (`ee`) и может быть опущен;
- о `--license` — параметр для указания лицензионного ключа Deckhouse Kubernetes Platform для аутентификации в официальном хранилище образов;
- о `<LICENSE_KEY>` — лицензионный ключ Deckhouse Kubernetes Platform;
- о `/home/user/d8-bundle` — директория, в которой будут расположены пакеты образов. Будет создана, если не существует.



Если загрузка образов будет прервана, повторный вызов команды продолжит загрузку, если с момента ее остановки прошло не более суток.

Пример команды для загрузки всех версий DKP EE, начиная с версии 1.59 (укажите лицензионный ключ):

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--since-version=1.59 /home/user/d8-bundle
```

Пример команды для загрузки актуальных версий DKP SE (укажите лицензионный ключ):

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--source='registry.deckhouse.ru/deckhouse/se' \
/home/user/d8-bundle
```

Пример команды для загрузки образов DKP из стороннего хранилища образов:

```
d8 mirror pull \
--source='corp.company.com:5000/sys/deckhouse' \
--source-login='<USER>' --source-password='<PASSWORD>' /home/user/d8-bundle
```

Пример команды для загрузки пакета баз данных сканера уязвимостей:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-modules /home/user/d8-bundle
```

Пример команды для загрузки пакетов всех доступных дополнительных модулей:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db /home/user/d8-bundle
```

Пример команды для загрузки пакетов модулей `stronghold` и `secrets-store-integration`:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold \
--include-module secrets-store-integration \
/home/user/d8-bundle
```

Пример команды для загрузки модуля `stronghold` с semver `^` ограничением от версии 1.2.0:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold@1.2.0 \
/home/user/d8-bundle
```

Пример команды для загрузки модуля `secrets-store-integration` с semver `~` ограничением от версии 1.1.0:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module secrets-store-integration@~1.1.0 \
/home/user/d8-bundle
```

Пример команды для загрузки точной версии модуля `stronghold` 1.2.5 с публикацией во все каналы релизов:

```
d8 mirror pull \
--license='<LICENSE_KEY>' \
--no-platform --no-security-db \
--include-module stronghold@v1.2.5 \
/home/user/d8-bundle
```

Другие параметры команды, доступные для использования:

- `--no-pull-resume` — принудительно начать загрузку сначала;
- `--no-platform` — пропустить загрузку пакета образов Deckhouse Kubernetes Platform (`platform.tar`);

- `--no-modules` — пропустить загрузку пакетов модулей (`module-*.tar`);
- `--no-security-db` — пропустить загрузку пакета баз данных сканера уязвимостей (`security.tar`);
- `--include-module / -i = name[@Major.Minor]` — загрузить определенный набор модулей по принципу белого списка (и, при необходимости, их минимальных версий). Укажите несколько раз, чтобы добавить в белый список больше модулей. Эти флаги игнорируются, если используются совместно с `--no-modules`.

Поддерживаются следующие синтаксисы для указания версий модулей:

- `module-name@1.3.0` — загрузка версий с `semver ^` ограничением (`^1.3.0`), включая `v1.3.0`, `v1.3.3`, `v1.4.1`;
- `module-name@~1.3.0` — загрузка версий с `semver ~` ограничением (`>=1.3.0 <1.4.0`), включая только `v1.3.0`, `v1.3.3`;
- `module-name@=v1.3.0` — загрузка точного соответствия тегу `v1.3.0`, публикация во все каналы релизов;
- `module-name@bobV1` — загрузка точного соответствия тегу `"bobV1"`, публикация во все каналы релизов.
- `--exclude-module / -e = name` — пропустить загрузку определенного набора модулей по принципу черного списка. Укажите несколько раз, чтобы добавить в черный список больше модулей. Игнорируется, если используются `--no-modules` или `--include-module`.
- `--modules-path-suffix` — изменить суффикс пути к репозиторию модулей в основном репозитории DKP. По умолчанию используется суффикс `/modules` (так, например, полный путь к репозиторию с модулями будет выглядеть как `registry.deckhouse.ru/deckhouse/EDITION/modules`);
- `--since-version=X.Y` — скачать все версии DKP, начиная с указанной минорной версии. Параметр будет проигнорирован, если указанная версия выше, чем версия на канале обновлений Rock Solid. Параметр не может быть использован одновременно с параметром `--deckhouse-tag`;
- `--deckhouse-tag` — скачать только конкретную версию DKP (без учета каналов обновлений). Параметр не может быть использован одновременно с параметром `--since-version`;
- `--gost-digest` — рассчитать контрольную сумму итогового набора образов DKP в формате ГОСТ Р 34.11-2012 (Стрибог). Контрольная сумма будет отображена и записана в файл с расширением `.tar.gostsum` в папке с TAR-архивом, содержащим образы DKP;
- `--source-login` и `--source-password` — данные для аутентификации в стороннем хранилище образов;
- `--images-bundle-chunk-size=N` — максимальный размер файлов (в ГБ), на которые нужно разбить архив образов. В результате работы вместо одного файла архива образов будет создан набор CHUNK-файлов (например, `d8.tar.NNNN.chunk`). Чтобы загрузить образы из такого набора файлов, укажите в команде `d8 mirror push` имя файла без суффикса `.NNNN.chunk` (например, `d8.tar` для файлов `d8.tar.NNNN.chunk`);
- `--tmp-dir` — путь к директории для временных файлов, который будет использоваться во время операций загрузки и выгрузки образов. Вся обработка выполняется в этом каталоге. Он должен иметь достаточный объем свободного дискового пространства, чтобы вместить весь загружаемый пакет образов. По умолчанию используется поддиректория `.tmp` в директории с пакетами образов.

Дополнительные параметры конфигурации для семейства команд `d8 mirror` доступны в виде переменных окружения:

- `HTTP_PROXY/HTTPS_PROXY` — URL-адрес прокси-сервера для запросов к HTTP(S)-хостам, которые не указаны в списке хостов в переменной `$NO_PROXY`;
- `NO_PROXY` — список хостов, разделенных запятыми, которые следует исключить из проксирования. Каждое значение может быть представлено в виде IP-адреса (`1.2.3.4`), CIDR (`1.2.3.4/8`), домена или символа (`*`). IP-адреса и домены также могут включать номер порта (`1.2.3.4:80`). Доменное имя соответствует как самому себе, так и всем поддоменам. Доменное имя, начинающееся с `.`, соответствует только поддоменам. Например, `foo.com` соответствует `foo.com` и `bar.foo.com`; `.y.com` соответствует `x.y.com`, но не соответствует `y.com`. Символ `*` отключает проксирование;
- `SSL_CERT_FILE` — путь до SSL-сертификата. Если переменная установлена, системные сертификаты не используются;
- `SSL_CERT_DIR` — список каталогов, разделенный двоеточиями. Определяет, в каких каталогах искать файлы SSL-сертификатов. Если переменная установлена, системные сертификаты не используются. Подробнее...
- `MIRROR_BYPASS_ACCESS_CHECKS` — установите для этого параметра значение `1`, чтобы отключить проверку корректности переданных учетных данных для хранилища образов.
- На хост с доступом к хранилищу образов, куда нужно загрузить образы DKP, скопируйте загруженный пакет образов DKP и установите Deckhouse CLI.
- Загрузите образы DKP в хранилище образов с помощью команды `d8 mirror push`.

Команда `d8 mirror push` загружает в хранилище образов образы из всех пакетов, которые присутствуют в переданной директории. При необходимости выгрузить в хранилище образов только часть пакетов вы можете либо выполнить команду для каждого необходимого пакета образов, передав ей прямой путь до TAR-пакета вместо директории, либо убрав расширение `.tar` у ненужных пакетов или переместив их вне директории.

Пример команды для загрузки пакетов образов из директории `/mnt/MEDIA/d8-images` (укажите данные для авторизации при необходимости):

```
d8 mirror push /mnt/MEDIA/d8-images 'corp.company.com:5000/sys/deckhouse' \
--registry-login='<USER>' --registry-password='<PASSWORD>'
```

Перед загрузкой образов убедитесь, что путь для загрузки в хранилище образов существует (в примере — `/sys/deckhouse`) и у используемой учетной записи есть права на запись.

Если вы используете Harbor, вы не сможете выгрузить образы в корень проекта. Используйте выделенный репозиторий в проекте для размещения образов DKP.

- После загрузки образов в хранилище образов можно переходить к установке DKP. Воспользуйтесь руководством по быстрому старту.

При запуске установщика используйте хранилище образов, в которое ранее были загружены образы, а не официальное публичное хранилище образов DKP. Для примера выше адрес запуска установщика будет иметь вид `corp.company.com:5000/sys/deckhouse/install:stable` вместо `registry.deckhouse.ru/deckhouse/ee/install:stable`.

В секции параметров `registry` `ModuleConfig` `deckhouse` при установке также используйте адрес вашего хранилища образов и данные авторизации (с версии DKP 1.75). Устаревший способ — использование `InitConfiguration` (параметры `imagesRepo`, `registryDockerCfg`).

Создание кластера и запуск DKP без использования каналов обновлений



Этот способ следует использовать только в случае, если в приватном хранилище нет образов, содержащих информацию о каналах обновлений.

Если необходимо установить DKP с отключенным автоматическим обновлением:

1. Используйте тег образа установщика соответствующей версии. Например, если вы хотите установить релиз `v1.44.3`, используйте образ `your.private.registry.com/deckhouse/install:v1.44.3`.
2. Укажите соответствующий номер версии в параметре `deckhouse.devBranch` в `InitConfiguration`.



Не указывайте параметр `deckhouse.releaseChannel` в `InitConfiguration`.

Если вы хотите отключить автоматические обновления для уже установленного DKP (включая обновления patch-релизов), удалите параметр `releaseChannel` из конфигурации модуля `deckhouse`.

Использование прокси-сервера

Пример шагов по настройке прокси-сервера на базе Squid...

1. Подготовьте сервер (или виртуальную машину). Сервер должен быть доступен с необходимых узлов кластера, и у него должен быть выход в интернет.
2. Установите Squid (здесь и далее примеры для Ubuntu):

```
apt-get install squid
```

3. Создайте файл конфигурации Squid:

```
cat <<EOF > /etc/squid/squid.conf
auth_param basic program /usr/lib/squid3/basic_ncsa_auth /etc/squid/passwords
auth_param basic realm proxy
acl authenticated proxy_auth REQUIRED
http_access allow authenticated

# Укажите необходимый порт. Порт 3128 используется по умолчанию.
http_port 3128
```

4. Создайте пользователя и пароль для аутентификации на прокси-сервере:

Пример для пользователя `test` с паролем `test` (обязательно измените):

```
echo "test:$(openssl passwd -crypt test)" >> /etc/squid/passwords
```

5. Запустите Squid и включите его автоматический запуск при загрузке сервера:

```
systemctl restart squid
systemctl enable squid
```

Для настройки DKP на работу с прокси-сервером используйте параметр `proxy` ресурса `ClusterConfiguration`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: OpenStack
  prefix: main
podSubnetCIDR: 10.111.0.0/16
serviceSubnetCIDR: 10.222.0.0/16
kubernetesVersion: "Automatic"
cri: "Containerd"
clusterDomain: "cluster.local"
proxy:
  httpProxy: "http://user:password@proxy.company.my:3128"
  httpsProxy: "https://user:password@proxy.company.my:8443"
```

Автозагрузка прокси-переменных пользователям в CLI

Начиная с версии 1.67, в DKP больше не настраивается файл `/etc/profile.d/d8-system-proxy.sh`, который ранее устанавливал прокси-переменные для пользователей. Для автозагрузки прокси-переменных пользователям в CLI используйте ресурс `NodeGroupConfiguration`:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: profile-proxy.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 99
  content: |
    {{- if .proxy }}
      {{- if .proxy.httpProxy }}
        export HTTP_PROXY={{ .proxy.httpProxy | quote }}
        export http_proxy=${HTTP_PROXY}
      {{- end }}
      {{- if .proxy.httpsProxy }}
        export HTTPS_PROXY={{ .proxy.httpsProxy | quote }}
        export https_proxy=${HTTPS_PROXY}
      {{- end }}
      {{- if .proxy.noProxy }}
        export NO_PROXY={{ .proxy.noProxy | join "," | quote }}
        export no_proxy=${NO_PROXY}
      {{- end }}
      bb-sync-file /etc/profile.d/profile-proxy.sh - << EOF
      export HTTP_PROXY=${HTTP_PROXY}
      export http_proxy=${HTTP_PROXY}
      export HTTPS_PROXY=${HTTPS_PROXY}
      export https_proxy=${HTTPS_PROXY}
      export NO_PROXY=${NO_PROXY}
      export no_proxy=${NO_PROXY}
      EOF
    {{- else }}
      rm -rf /etc/profile.d/profile-proxy.sh
    {{- end }}
```

Дополнительные ресурсы

- Быстрый старт
- Поддерживаемые версии ОС и Kubernetes
- Интеграция с инфраструктурой
- Установка DKP в закрытом окружении
- Подготовка к Production
- Глобальные кастомные ресурсы
- Глобальные параметры

Как настроить?

Основы конфигурации Deckhouse

Deckhouse конфигурируется с помощью:

- **Глобальных настроек.** Глобальные настройки хранятся в ресурсе `ModuleConfig/global`. Эти настройки можно рассматривать как специальный модуль `global`, который нельзя отключить.
- **Настроек модулей.** Настройки каждого модуля хранятся в ресурсе `ModuleConfig`, имя которого совпадает с именем модуля (в kebab-case).
- **Кастомных ресурсов.** Некоторые модули настраиваются с помощью дополнительных кастомных ресурсов.

Пример набора кастомных ресурсов конфигурации Deckhouse:

```
# Глобальные настройки.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 1
  settings:
    modules:
      publicDomainTemplate: "%s.kube.company.my"
---
# Настройки модуля monitoring-ping.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  settings:
    externalTargets:
      - host: 8.8.8.8
---
# Отключить модуль dashboard.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: dashboard
spec:
  enabled: false
```

Посмотреть список кастомных ресурсов `ModuleConfig`, состояние модулей (включен/выключен) и их статус можно с помощью следующей команды:

```
d8 k get moduleconfigs
```

Пример вывода...

```
$ d8 k get moduleconfigs
NAME           ENABLED  VERSION  AGE      MESSAGE
deckhouse      true     1         12h
documentation  true     1         12h
global         1        12h
prometheus     true     2         12h
upmeter        false    2         12h
```

Чтобы изменить глобальную конфигурацию Deckhouse или конфигурацию модуля, нужно создать или отредактировать соответствующий ресурс `ModuleConfig`.

Например, чтобы отредактировать конфигурацию модуля `upmeter`, выполните следующую команду:

```
d8 k edit moduleconfig/upmeter
```

После завершения редактирования изменения применяются автоматически.

Изменение конфигурации кластера



Для применения изменений конфигурации узлов необходимо выполнить команду `dhctl converge`, запустив инсталлятор DKP. Эта команда синхронизирует состояние узлов с указанным в конфигурации.

Общие параметры кластера хранятся в структуре `ClusterConfiguration`.

Чтобы изменить общие параметры кластера, выполните команду:

```
d8 system edit cluster-configuration
```

После сохранения изменений DKP автоматически приведёт кластер в соответствие с новой конфигурацией. В зависимости от размеров кластера это может занять некоторое время.

Изменение защищённых параметров

Некоторые параметры кластера критичны для его работы и по умолчанию не могут быть изменены в работающем кластере. К таким параметрам относятся:

- `podSubnetCIDR` — адресное пространство сети подов;
- `podSubnetNodeCIDRPrefix` — размер префикса сети подов на узел;
- `serviceSubnetCIDR` — адресное пространство сети сервисов.

Попытки изменить эти параметры будут заблокированы admission webhook с сообщением об ошибке.



Изменение этих параметров в работающем кластере может привести к:

- полной потере доступа к Kubernetes API;
- недействительности TLS-сертификатов;
- необходимости перезапуска всех узлов кластера и компонентов control plane;
- несогласованности данных при прерывании процесса.

Рекомендуется пересоздать кластер вместо изменения этих параметров.

Если в изменении этих параметров есть необходимость (например, для тестирования или в исключительных случаях), можно обойти механизм защиты.

Изменение защищённых параметров с использованием dhctl

Используйте утилиту `dhctl` из контейнера инсталлятора DKP с флагом `--yes-i-am-sane-and-i-understand-what-i-am-doing`.

Она автоматически:

- добавит аннотацию `deckhouse.io/allow-unsafe` к секреты `d8-cluster-configuration`;
- откроет редактор для изменения конфигурации;
- удалит аннотацию после сохранения изменений.

Для изменения защищённых параметров с помощью `dhctl` выполните следующие шаги:

1. Получите текущую версию и редакцию DKP из вашего кластера. Для этого **на локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии:

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]')
```

2. Запустите контейнер инсталлятора DKP (при необходимости измените адрес registry):

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

Где `<MOUNT_OPTIONS>` — параметры монтирования файлов в контейнер установщика, такие как:

- SSH-ключи доступа;
- файл конфигурации;
- файл ресурсов и т. д.

3. Внутри контейнера выполните следующую команду для редактирования конфигурации кластера:

```
dhctl config edit cluster-configuration \
--ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
--ssh-user=<USERNAME> \
--ssh-host=<MASTER-NODE-HOST> \
--yes-i-am-sane-and-i-understand-what-i-am-doing
```

Где:

- `<SSH_KEY_FILENAME>` — имя файла вашего приватного SSH-ключа,
- `<USERNAME>` — SSH-пользователь с правами `sudo` на целевом master-узле кластера,
- `<MASTER-NODE-HOST>` — IP-адрес или имя хоста master-узла.

4. Отредактируйте конфигурацию в открывшемся редакторе, сохраните изменения и выйдите из редактора.



Даже с обходом механизма защиты **нет гарантии**, что кластер продолжит корректно функционировать после изменения этих параметров. Будьте готовы к возможности полного отказа кластера и имейте резервный план.

Изменение защищённых параметров вручную

Если необходимо вручную отредактировать конфигурацию:

1. Добавьте аннотацию `deckhouse.io/allow-unsafe` к секреты `d8-cluster-configuration`:

```
d8 k -n kube-system annotate secret d8-cluster-configuration deckhouse.io/allow-unsafe="true"
```

- Получите текущую конфигурацию, декодируйте её и сохраните в файл:

```
d8 k -n kube-system get secret d8-cluster-configuration \
-o jsonpath='{.data.cluster-configuration\.yaml}' | base64 -d > cluster-config.yaml
```

- Отредактируйте файл `cluster-config.yaml` в предпочитаемом редакторе:

```
vi cluster-config.yaml
```

- Закодируйте отредактированную конфигурацию и обновите секрет:

```
d8 k -n kube-system patch secret d8-cluster-configuration \
--patch="{\"data\":{\"cluster-configuration.yaml\": \"$(base64 -w0 < cluster-config.yaml)\"}}"
```

- Удалите аннотацию после применения изменений:

```
d8 k -n kube-system annotate secret d8-cluster-configuration deckhouse.io/allow-unsafe-
```

- Удалите временный файл с новой конфигурацией:

```
rm cluster-config.yaml
```



Если вы забудете удалить аннотацию `deckhouse.io/allow-unsafe`, механизм защиты останется отключённым, оставляя ваш кластер уязвимым для случайных изменений конфигурации.

Просмотр текущих настроек

DKP управляется с помощью глобальных настроек, настроек модулей и различных кастомных ресурсов.

- Для просмотра глобальных настроек выполните:

```
d8 k get mc global -o yaml
```

- Для просмотра состояния всех модулей (доступно для Deckhouse версии 1.47+):

```
d8 k get modules
```

- Для просмотра настроек модуля `user-authn`:

```
d8 k get moduleconfigs user-authn -o yaml
```

Настройка модуля

Модуль настраивается с помощью ресурса `ModuleConfig`, имя которого совпадает с именем модуля (в kebab-case). `ModuleConfig` имеет следующие поля:

- `metadata.name` — название модуля Deckhouse в kebab-case (например, `prometheus`, `node-manager`);
- `spec.version` — версия схемы настроек модуля (целое число, больше нуля). Обязательное поле, если `spec.settings` не пустое. Номер актуальной версии можно увидеть в документации модуля в разделе *Настройки*:
 - Deckhouse поддерживает обратную совместимость версий схемы настроек модуля. Если используется схема настроек устаревшей версии, при редактировании или просмотре кастомного ресурса будет выведено предупреждение о необходимости обновить схему настроек модуля;
- `spec.settings` — настройки модуля. Необязательное поле, если используется поле `spec.enabled`. Описание возможных настроек можно найти в документации модуля в разделе *Настройки*;
- `spec.enabled` — необязательное поле для явного включения или отключения модуля. Если не задано, модуль может быть включен по умолчанию в одном из наборов модулей.



Deckhouse не изменяет кастомные ресурсы `ModuleConfig`. Это позволяет применять подход Infrastructure as Code (IaC) при хранении конфигурации. Другими словами, можно воспользоваться всеми преимуществами системы контроля версий для хранения настроек Deckhouse, использовать Helm, `d8 k` и другие привычные инструменты.

Пример кастомного ресурса для настройки модуля `kube-dns`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  settings:
    stubZones:
      - upstreamNameservers:
          - 192.168.121.55
          - 10.2.7.80
        zone: directory.company.my
    upstreamNameservers:
      - 10.2.100.55
      - 10.2.200.55

```

Некоторые модули настраиваются с помощью дополнительных ресурсов. Воспользуйтесь поиском (вверху страницы) или выберите модуль в меню слева, чтобы просмотреть документацию по его настройкам и используемым кастомным ресурсам.

Включение и отключение модуля

i Некоторые модули могут быть включены по умолчанию в зависимости от используемого набора модулей.

Для явного включения или отключения модуля необходимо установить `true` или `false` в поле `.spec.enabled` в соответствующем кастомном ресурсе `ModuleConfig`. Если для модуля нет такого кастомного ресурса `ModuleConfig`, его нужно создать.

Пример явного выключения модуля `user-authn` (модуль будет выключен независимо от используемого набора модулей):

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  enabled: false

```

Проверить состояние модуля можно с помощью команды `d8 k get moduleconfig <ИМЯ_МОДУЛЯ>`.

Пример:

```

$ d8 k get moduleconfig user-authn
NAME          ENABLED  VERSION  AGE   MESSAGE
user-authn    false   1        12h

```

Наборы модулей

В зависимости от используемого набора модулей (bundle) модули могут быть включены или выключены по умолчанию.

⚠ Таблица ниже описывает наборы модулей только для встроенных модулей Deckhouse Kubernetes Platform. Модули, подключаемые из источника модулей, в этой таблице не учитываются.

Набор модулей (bundle)	Список включенных по умолчанию модулей
------------------------	--

Набор модулей (bundle)	Список включенных по умолчанию модулей
Default	• node-local-dns • admission-policy-engine • cert-manager • chrony • control-plane-manager • dashboard • deckhouse • documentation • deckhouse-tools • descheduler • extended-monitoring • ingress-nginx • kube-dns • kube-proxy • local-path-provisioner • log-shipper • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes-control-plane • monitoring-kubernetes • monitoring-ping • multitenancy-manager • namespace-configurator • registry-packages-proxy • node-manager • operator-prometheus • prometheus • prometheus-metrics-adapter • secret-copier • terraform-manager • upmeter • user-authn • user-authz • vertical-pod-autoscaler • registry
Managed	• admission-policy-engine • cert-manager • dashboard • deckhouse • documentation • deckhouse-tools • descheduler • extended-monitoring • ingress-nginx • local-path-provisioner • log-shipper • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-ping • multitenancy-manager • namespace-configurator • operator-prometheus • prometheus • prometheus-metrics-adapter • secret-copier • upmeter • user-authz • vertical-pod-autoscaler
Minimal	• deckhouse

Особенности работы с набором модулей Minimal



Обратите внимание, что в наборе модулей `Minimal` не включен ряд базовых модулей (например, модуль работы с CNI).

Deckhouse Kubernetes Platform с набором модулей `Minimal` без включения базовых модулей сможет работать только в уже развернутом кластере.

Для установки DKP с набором модулей `Minimal` включите как минимум следующие модули, указав их в конфигурационном файле установщика:

- `cni-cilium` или другой модуль управления CNI (при необходимости);
- `control-plane-manager`;

- kube-dns;
- terraform-manager, в случае развертывания облачного кластера;
- node-manager;
- registry-packages-proxy;
- модуль облачного провайдера (например, cloud-provider-aws для AWS), в случае развертывания облачного кластера.

Доступ к документации текущей версии

Документация запущенной в кластере версии Deckhouse доступна по адресу `documentation.<cluster domain>`, где `<cluster domain>` — DNS-имя в соответствии с шаблоном из параметра `modules.publicDomainTemplate` глобальной конфигурации.



Документация доступна, если в кластере включен модуль `documentation`. Он включен по умолчанию, кроме варианта поставки `Minimal`.

Управление размещением компонентов Deckhouse

Выделение узлов под определенный вид нагрузки

Если в параметрах модуля не указаны явные значения `nodeSelector/tolerations`, то для всех модулей используется следующая стратегия:

1. Если параметр `nodeSelector` модуля не указан, то Deckhouse попытается вычислить `nodeSelector` автоматически. В этом случае, если в кластере присутствуют узлы с лейблами из списка или лейблами определенного формата, Deckhouse укажет их в качестве `nodeSelector` ресурсам модуля.
2. Если параметр `tolerations` модуля не указан, то подам модуля автоматически устанавливаются все возможные `toleration`'ы, (подробнее).
3. Отключить автоматическое вычисление параметров `nodeSelector` или `tolerations` можно, указав значение `false`.
4. При отсутствии в кластере выделенных узлов и автоматическом выборе `nodeSelector`, как описано в пункте 1, `nodeSelector` в ресурсах модуля указан не будет. Модуль в таком случае будет использовать любой узел с не конфликтующими `taints`.

Возможность настройки `nodeSelector` и `tolerations` отключена для модулей:

- которые работают на всех узлах кластера (например, `cni-flannel`, `monitoring-ping`);
- которые работают на всех master-узлах (например, `prometheus-metrics-adapter`, `vertical-pod-autoscaler`).

Особенности автоматики, зависящие от типа модуля



Ниже описана базовая (общая) логика автоматического выбора узлов для размещения компонентов модулей, когда в настройках модуля не заданы явные значения `nodeSelector` и `tolerations`. Некоторые модули могут дополнять или изменять эту логику (например, использовать механизмы Kubernetes, такие как `affinity/anti-affinity`, `topologySpreadConstraints`, или собственные правила выбора узлов). Подробности приведены в документации соответствующего модуля.

Если `nodeSelector` не задан явно в конфигурации модуля, Deckhouse определяет размещение компонентов автоматически. Порядок выбора узлов зависит от типа модуля.

- Модуль `console`:
 - Если `nodeSelector` не задан явно в конфигурации модуля, компоненты модуля размещаются на `control-plane`-узлах.
 - Если `control-plane`-узлы недоступны для такого размещения, используются узлы с лейблом `node-role.deckhouse.io/system`.
- Модули `monitoring` (`operator-prometheus`, `prometheus` и `vertical-pod-autoscaler`):
 - Порядок поиска узлов (для определения `nodeSelector`):
 1. Наличие узла с лейблом `node-role.deckhouse.io/MODULE NAME`.
 2. Наличие узла с лейблом `node-role.deckhouse.io/monitoring`.
 3. Наличие узла с лейблом `node-role.deckhouse.io/system`.
 - Добавляемые `toleration`'ы (добавляются одновременно все):
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "MODULE NAME"}` (например, `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "operator-prometheus"}`);
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "monitoring"}`;
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "system"}`.
- Модули `frontend` (исключительно модуль `ingress-nginx`):
 - Порядок поиска узлов (для определения `nodeSelector`):
 1. Наличие узла с лейблом `node-role.deckhouse.io/MODULE NAME`.
 2. Наличие узла с лейблом `node-role.deckhouse.io/frontend`.
 - Добавляемые `toleration`'ы (добавляются одновременно все):
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "MODULE NAME"}`;
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "frontend"}`.
- Все остальные модули:
 - Порядок поиска узлов (для определения `nodeSelector`):
 1. Наличие узла с лейблом `node-role.deckhouse.io/MODULE_NAME` (например, `node-role.deckhouse.io/cert-manager`).
 2. Наличие узла с лейблом `node-role.deckhouse.io/system`.
 - Добавляемые `toleration`'ы (добавляются одновременно все):
 - `{"key": "dedicated.deckhouse.io", "operator": "Equal", "value": "MODULE_NAME"}` (например,

- `{ "key": "dedicated.deckhouse.io", "operator": "Equal", "value": "network-gateway" } );`
`{ "key": "dedicated.deckhouse.io", "operator": "Equal", "value": "system" } }.`

Дополнительные ресурсы

- Глобальные параметры
- Глобальные кастомные ресурсы
- Глобальные кастомные ресурсы
- Параметры модуля `deckhouse`
- Документация модуля `kube-dns`
- Документация модуля `user-authn`

Работа с container registries и редакциями

В этом разделе описывается работа с container registries (registry) и редакциями в Deckhouse Kubernetes Platform.

Работа с registry

В разделе рассматривается работа с registry в функционирующем кластере. Если вас интересует информация про работу с registry при установке кластера, перейдите в раздел «Установка платформы».

В разделе «Сторонний registry» описан процесс переключения работающего кластера DKP на использование стороннего registry. В разделе «Внутренний registry» рассматривается подготовка к переключению между режимами работы кластера: с использованием внутреннего container registry или без использования, а также процессы переключения. В разделе «Восстановление подключения к registry» описан процесс восстановления загрузки образов Deckhouse Kubernetes Platform при истекшем лицензионном токене.

Работа с редакциями

В разделе «Переключение редакций» описаны возможные варианты переключения редакций в работающем кластере Deckhouse Kubernetes Platform.

Переключение работающего кластера DKP на использование стороннего registry

⚠ При использовании модуля `registry` смена адреса и параметров registry выполняется в секции `registry` конфигурации модуля `deckhouse`. Пример настройки приведен в документации модуля `registry`.

⚠ Использование registries, отличных от `registry.deckhouse.io` и `registry.deckhouse.ru`, доступно только в коммерческих редакциях Deckhouse Kubernetes Platform.

⚠ Если в процессе переключения образ какого-либо модуля не загрузился заново и модуль не переустановился, для устранения проблемы воспользуйтесь инструкцией.

Для переключения кластера на использование стороннего registry выполните следующие действия:

1. Выполните команду `deckhouse-controller helper change-registry` из пода DKP с параметрами нового registry. Пример запуска:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller helper change-registry --user MY-USER --password MY-PASSWORD registry.example.com/deckhouse/ee
```

2. Если registry использует самоподписанные сертификаты, положите корневой сертификат соответствующего сертификата registry в файл `/tmp/ca.crt` в поде DKP и добавьте к вызову опцию `--ca-file /tmp/ca.crt` или вставьте содержимое CA в переменную, как в примере ниже:

```
CA_CONTENT=$(cat <<EOF
-----BEGIN CERTIFICATE-----
CERTIFICATE
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
CERTIFICATE
-----END CERTIFICATE-----
EOF
)
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- bash -c "echo '$CA_CONTENT' > /tmp/ca.crt && deckhouse-controller helper change-registry --ca-file /tmp/ca.crt --user MY-USER --password MY-PASSWORD registry.example.com/deckhouse/ee"
```

Просмотреть список доступных ключей команды `deckhouse-controller helper change-registry` можно, выполнив следующую команду:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller helper change-registry --help
```

Пример вывода:

```
usage: deckhouse-controller helper change-registry [<flags>] <new-registry>

Change registry for deckhouse images.

Flags:
  --help                Show context-sensitive help (also try --help-long and --help-man).
  --user=USER           User with pull access to registry.
  --password=PASSWORD   Password/token for registry user.
  --ca-file=CA-FILE     Path to registry CA.
  --scheme=SCHEME       Used scheme while connecting to registry, http or https.
  --dry-run             Don't change deckhouse resources, only print them.
  --new-deckhouse-tag=NEW-DECKHOUSE-TAG
                        New tag that will be used for deckhouse deployment image (by default
                        current tag from deckhouse deployment will be used).

Args:
  <new-registry>       Registry that will be used for deckhouse images (example:
                        registry.deckhouse.io/deckhouse/ce). By default, https will be used, if you
                        need
                        http - provide '--scheme' flag with http value
```

3. Дождитесь перехода пода registry в статус `Ready`. Если под находится в статусе `ImagePullBackoff`, перезапустите его.
4. Дождитесь применения новых настроек на master-узле.

Проверьте журнал системного сервиса bashible на master-узле, например, с помощью следующей команды:

```
journalctl -u bashible -n 20
```

В журнале должно появиться сообщение `Configuration is in sync, nothing to do.`

Пример вывода при просмотре журнала сервиса bashible:

```
$ journalctl -u bashible -n 20
...
Aug 13 05:03:08 kube-master-0 systemd[1]: Started Bashible service.
Aug 13 05:03:10 kube-master-0 bash[1847265]: Configuration is in sync, nothing to do. <--
Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Deactivated successfully.
Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Consumed 1.075s CPU time.
```

5. Если необходимо отключить автоматическое обновление registry через сторонний registry, удалите параметр `releaseChannel` из конфигурации модуля `deckhouse`.
6. Проверьте, не осталось ли в кластере подов с оригинальным адресом registry:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[]
| select(.image | startswith("registry.deckhouse")) | .metadata.namespace + "\t" +
.metadata.name' | sort | uniq
```

Дополнительные ресурсы

- Параметры модуля `deckhouse`
- Документация модуля `registry`

Восстановление подключения к registry при истекшем и неверном лицензионном токене



Используйте эту инструкцию только если недоступна штатная процедура смены registry.

Если после истечения лицензионного токена поды Deckhouse Kubernetes Platform (DKP) были перезапущены, в их логах появится ошибка подключения к registry при загрузке образов DKP. Чтобы переключить кластер на новый токен, на любом master-узле выполните следующие шаги:

1. Загрузите текущую конфигурацию секрета `deckhouse-registry` во временный файл:

```
d8 k -n d8-system get secret deckhouse-registry -o yaml > /tmp/deckhouse-registry.yaml
```

2. Во временном файле `/tmp/deckhouse-registry.yaml` замените значение поля `.dockerconfigjson` на Base64-

кодированную строку с параметрами подключения к registry. Получить нужную строку можно командами ниже, подставив свои значения MYPASSWORD и MYREGISTRY:

```
declare MYUSER='license-token'
declare MYPASSWORD='example-token'
declare MYREGISTRY='example-registry.deckhouse.ru'
MYAUTH=$(echo -n "$MYUSER:$MYPASSWORD" | base64 -w0)
MYRESULTSTRING=$(echo -n "{\"auths\":{\"$MYREGISTRY\":{\"username\":\"$MYUSER\",\"password\":\"$MYPASSWORD\",\"auth\":\"$MYAUTH\"}}}" | base64 -w0)
echo "$MYRESULTSTRING"
```

3. Разрешите изменение устаревшего секрета:

```
d8 k delete validatingadmissionpolicybindings.admissionregistration.k8s.io heritage-label-objects.deckhouse.io
```

4. Импортируйте обновленную конфигурацию:

```
d8 k -n d8-system apply -f /tmp/deckhouse-registry.yaml
```

5. Найдите проблемный под `deckhouse` на текущем master-узле и удалите его:

```
d8 k get pods -n d8-system -O wide
d8 k delete pod -n d8-system deckhouse-<id>
```

6. Убедитесь, что новый под `deckhouse` запустился корректно:

```
d8 k get pods -n d8-system
```

7. При необходимости удалите остальные поды `deckhouse`, находящиеся в некорректном статусе.

8. Повторите штатную процедуру смены registry, подставив ваш токен и нужный адрес registry и редакцию вместо `example`:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller helper change-registry --user license-token --password MY-PASSWORD registry-example.deckhouse.ru/deckhouse/example
```

Переключение редакций

Переключение DKP с EE на CSE

Переключение DKP с EE на CSE может быть выполнено одним из следующих способов:

- с использованием модуля `registry`;
- без модуля `registry`.



Инструкция подразумевает использование публичного адреса container registry: `registry-cse.deckhouse.ru`.

В DKP CSE не поддерживается работа облачных кластеров и некоторых модулей. Подробнее о поддерживаемых модулях можно узнать на странице сравнения редакций.

Миграция на DKP CSE возможна только с версии DKP EE 1.58, 1.64 или 1.67.

Актуальные версии DKP CSE: 1.58.2 для релиза 1.58, 1.64.1 для релиза 1.64 и 1.67.0 для релиза 1.67. Эти версии потребуются использовать далее для указания переменной `DECKHOUSE_VERSION`.

Переход поддерживается только между одинаковыми минорными версиями, например, с DKP EE 1.64 на DKP CSE 1.64. Переход с версии EE 1.58 на CSE 1.67 потребует промежуточной миграции: сначала на EE 1.64, затем на EE 1.67, и только после этого — на CSE 1.67. Попытки обновить версию на несколько релизов сразу могут привести к неработоспособности кластера.

Deckhouse CSE 1.58 и 1.64 поддерживает Kubernetes версии 1.27, DKP CSE 1.67 поддерживает Kubernetes версий 1.27 и 1.29.

При переключении на DKP CSE возможна временная недоступность компонентов кластера.

Если в процессе переключения образ какого-либо модуля не загрузился заново и модуль не переустановился, для устранения проблемы воспользуйтесь инструкцией.

Для переключения кластера Deckhouse Enterprise Edition на Certified Security Edition нужным способом выполните описанные ниже действия (все команды выполняются на master-узле кластера от имени пользователя с настроенным контекстом `kubect1` или от имени суперпользователя).

Переключение с использованием модуля `registry`

1. Убедитесь, что кластер был переключен на использование модуля `registry`. Если модуль не задействован, перейдите к инструкции по переключению без модуля `registry`.

- Настройте кластер на использование необходимой версии Kubernetes (информация о версии приведена в разделе Переключение DKP с EE на CSE). Для этого:

- Выполните команду:

```
d8 platform edit cluster-configuration
```

- Измените параметр `kubernetesVersion` на необходимое значение, например, `"1.27"` (в кавычках) для Kubernetes 1.27.
- Сохраните изменения. Узлы кластера начнут последовательно обновляться.
- Дождитесь окончания обновления. Отслеживать ход обновления можно с помощью команды `d8 k get po`. Обновление можно считать завершенным, когда в выводе команды у каждого узла кластера в колонке `VERSION` появится обновленная версия.

- Подготовьте переменные с лицензионным ключом:

```
LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
```

- Запустите временный под Deckhouse новой редакции, чтобы получить актуальные дайджесты и список модулей:

```
d8 k create secret docker-registry cse-image-pull-secret \
--docker-server=registry-cse.deckhouse.ru \
--docker-username=license-token \
--docker-password=${LICENSE_TOKEN}

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print $NF}')
d8 k run cse-image \
--image=registry-cse.deckhouse.ru/deckhouse/cse/install:$DECKHOUSE_VERSION \
--overrides="{\"spec\": {\"imagePullSecrets\": [{\"name\": \"cse-image-pull-secret\"}]}}" \
--command sleep -- infinity
```

Как только под перейдет в статус `Running`, выполните следующие команды:

```
CSE_MODULES=$(d8 k exec cse-image -- ls -l deckhouse/modules/ | awk {'print $9'} | grep -oP "\d.*-w*" |
cut -c5-)
USED_MODULES=$(d8 k get modules -O custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -Fv -f <(echo $CSE_MODULES | tr ' ' '\n'))
```

- Убедитесь, что используемые в кластере модули поддерживаются в Deckhouse CSE. Например, в Deckhouse CSE 1.58 и 1.64 отсутствует модуль `cert-manager`. Поэтому, перед отключением модуля `cert-manager` необходимо перевести режим работы HTTPS некоторых компонентов (например `user-authn` или `prometheus`) на альтернативные варианты работы, либо изменить глобальный параметр, отвечающий за режим работы HTTPS в кластере.

Отобразить список модулей, которые не поддерживаются в Deckhouse CSE и будут отключены, можно следующей командой:

```
echo $MODULES_WILL_DISABLE
```



Проверьте список и убедитесь, что функциональность указанных модулей не задействована в кластере, и вы готовы к их отключению.

Отключите неподдерживаемые в Deckhouse CSE модули:

```
echo $MODULES_WILL_DISABLE |
tr ' ' '\n' | awk {'print "d8 k -n d8-system exec deploy/deckhouse -- deckhouse-controller
module disable",$1'} | bash
```

В Deckhouse CSE не поддерживается компонент `earlyOOM`. Отключите его с помощью настройки.

Дождитесь перехода пода Deckhouse в статус `Ready` и выполнения всех задач в очереди.

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller queue list
```

Проверьте, что отключенные модули перешли в состояние `Disabled`.

```
d8 k get modules
```

- Удалите созданный секрет и под:

```
d8 k delete pod/cse-image
d8 k delete secret/cse-image-pull-secret
```

7. Выполните переключение на новую редакцию. Для этого укажите следующие параметры в ModuleConfig `deckhouse` (для подробной настройки ознакомьтесь с конфигурацией модуля `deckhouse`):

```
---
# Пример для Direct-режима.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        # Relax mode используется для проверки наличия текущей версии Deckhouse в указанном
        registry.
        # Для переключения между редакциями необходимо использовать данный режим проверки
        registry.
        checkMode: Relax
        imagesRepo: registry-cse.deckhouse.ru/deckhouse/cse
        scheme: HTTPS
        # Укажите свой параметр <LICENSE_TOKEN>.
        license: <LICENSE_TOKEN>
---
# Пример для Unmanaged-режима.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        # Relax mode используется для проверки наличия текущей версии Deckhouse в указанном
        registry.
        # Для переключения между редакциями необходимо использовать данный режим проверки.
        checkMode: Relax
        imagesRepo: registry-cse.deckhouse.ru/deckhouse/cse
        scheme: HTTPS
        # Укажите свой параметр <LICENSE_TOKEN>.
        license: <LICENSE_TOKEN>
```

8. Дождитесь переключения `registry`. Для проверки выполнения переключения воспользуйтесь инструкцией.

Пример вывода:

```
conditions:
- lastTransitionTime: "...
  message: |-
    Mode: Relax
    registry-cse.deckhouse.ru: all 1 items are checked
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
```

9. После переключения, удалите из ModuleConfig `deckhouse` параметр `checkMode: Relax`, чтобы активировать выполнение проверки по умолчанию. Удаление запустит проверку наличия критически важных компонентов в `registry`.
10. Дождитесь выполнения проверки. Статус переключения режима `registry` можно получить, воспользовавшись

инструкцией.

Пример вывода:

```
conditions:
- lastTransitionTime: "...
  message: |-
    Mode: Default
    registry-cse.deckhouse.ru: all 155 items are checked
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
```

11. Проверьте, не осталось ли в кластере подов с адресом registry для Deckhouse EE:

Для Unmanaged-режима:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[]
| select(.image | contains("deckhouse.ru/deckhouse/ee"))) | .metadata.namespace + "\t" +
.metadata.name' | sort | uniq
```

Для других режимов, использующих фиксированный адрес (данная проверка не учитывает внешние модули):

```
# Получаем список актуальных дайджестов из файла images_digests.json внутри Deckhouse.
IMAGES_DIGESTS=$(d8 k -n d8-system exec -i svc/deckhouse-leader -C deckhouse -- cat
/deckhouse/modules/images_digests.json | jq -r '[][]' | sort -u)

# Проверяем, есть ли поды, использующие образы Deckhouse по адресу `registry.d8-
system.svc:5001/system/deckhouse`
# с дайджестом, отсутствующим в списке актуальных дайджестов из IMAGES_DIGESTS.
d8 k get pods -A -o json |
jq -r --argjson digests "$(printf '%s\n' $IMAGES_DIGESTS | jq -R . | jq -s .)" '
.items[]
| {name: .metadata.name, namespace: .metadata.namespace, containers: .spec.containers}
| select(.containers != null)
| select(
  .containers[]
  | select(.image | test("registry.d8-system.svc:5001/system/deckhouse") and
test("@sha256:"))
  | .image as $img
  | ($img | split("@") | last) as $digest
  | ($digest | IN($digests[]) | not)
)
| .namespace + "\t" + .name
' | sort -u
```

Если в выводе присутствуют поды модуля `chrony`, заново включите данный модуль (в Deckhouse CSE этот модуль по умолчанию выключен):

```
d8 k -n d8-system exec deploy/deckhouse -- deckhouse-controller module enable chrony
```

Переключение без модуля registry

1. Если модуль `registry` включен, отключите его с помощью инструкции.
2. Настройте кластер на использование необходимой версии Kubernetes (информация о версииности приведена в разделе Переключение DKP с EE на CSE). Для этого:

- Выполните команду:

```
d8 platform edit cluster-configuration
```

- Измените параметр `kubernetesVersion` на необходимое значение, например, `"1.27"` (в кавычках) для Kubernetes 1.27.
- Сохраните изменения. Узлы кластера начнут последовательно обновляться.
- Дождитесь окончания обновления. Отслеживать ход обновления можно с помощью команды `d8 k get po`. Обновление можно считать завершенным, когда в выводе команды у каждого узла кластера в колонке `VERSION` появится обновленная версия.

3. Подготовьте переменные с лицензионным ключом и создайте ресурс `NodeGroupConfiguration` для переходной

авторизации в `registry-cse.deckhouse.ru`:

i Перед созданием ресурса ознакомьтесь с разделом [Как добавить конфигурацию для дополнительного registry](#)

```
LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
d8 k apply -f - <<EOF
---
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-cse-config.sh
spec:
  nodeGroups:
    - '*'
  bundles:
    - '*'
  weight: 30
  content: |
    _on_containerd_config_changed() {
      bb-flag-set containerd-need-restart
    }
    bb-event-on 'containerd-config-file-changed' '_on_containerd_config_changed'

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/cse-registry.toml - containerd-config-file-changed <<
"EOF_TOML"
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry-cse.deckhouse.ru"]
          endpoint = ["https://registry-cse.deckhouse.ru"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."registry-
cse.deckhouse.ru".auth]
          auth = "$AUTH_STRING"
    EOF_TOML
EOF
```

Дождитесь завершения синхронизации и появления файла `/etc/containerd/conf.d/cse-registry.toml` на узлах.

Статус синхронизации можно отследить по значению `UPTODATE` (отображаемое число узлов в этом статусе должно совпадать с общим числом узлов (`NODES`) в группе):

```
d8 k get ng -O custom-columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate -W
```

Пример вывода:

NAME	NODES	READY	UPTODATE
master	1	1	1
worker	2	2	2

В журнале `systemd`-сервиса `bashible` должно появиться сообщение `Configuration is in sync, nothing to do` в результате выполнения следующей команды:

```
journalctl -u bashible -n 5
```

Пример вывода:

```
Aug 21 11:04:28 master-ee-to-cse-0 bashible.sh[53407]: Configuration is in sync, nothing to do.
Aug 21 11:04:28 master-ee-to-cse-0 bashible.sh[53407]: Annotate node master-ee-to-cse-0 with
annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-cse-0 bashible.sh[53407]: Successful annotate node master-ee-to-
cse-0 with annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-cse-0 systemd[1]: bashible.service: Deactivated successfully.
```

4. Выполните следующие команды для запуска временного пода Deckhouse CSE для получения актуальных

дайджестов и списка модулей:

```
DECKHOUSE_VERSION=v<ВЕРСИЯ_DECKHOUSE_CSE>
# Например, DECKHOUSE_VERSION=v1.58.2
d8 k run cse-image --image=registry-cse.deckhouse.ru/deckhouse/cse/install:$DECKHOUSE_VERSION --command sleep
-- infinity
```

Как только под перейдёт в статус **Running**, выполните следующие команды:

```
CSE_SANDBOX_IMAGE=$(d8 k exec cse-image -- cat deckhouse/candi/images_digests.json | grep pause | grep -oE
'sha256:\w*')
CSE_K8S_API_PROXY=$(d8 k exec cse-image -- cat deckhouse/candi/images_digests.json | grep kubernetesApiProxy |
grep -oE 'sha256:\w*')
CSE_MODULES=$(d8 k exec cse-image -- ls -l deckhouse/modules/ | awk {'print $9'} | grep -oP "\d.*-\w*" |
cut -c5-)
USED_MODULES=$(d8 k get modules -O custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -Fvx -f <(echo $CSE_MODULES | tr ' '
'\n'))
CSE_DECKHOUSE_KUBE_RBAC_PROXY=$(d8 k exec cse-image -- cat deckhouse/candi/images_digests.json | jq -r
".common.kubeRbacProxy")
```

i Дополнительная команда, которая необходима только при переключении на Deckhouse CSE версии 1.64:

```
CSE_DECKHOUSE_INIT_CONTAINER=$(d8 k exec cse-image -- cat deckhouse/candi/images_digests.json | jq -r
".common.init")
```

- Убедитесь, что используемые в кластере модули поддерживаются в Deckhouse CSE. Например, в Deckhouse CSE 1.58 и 1.64 отсутствует модуль `cert-manager`. Поэтому, перед отключением модуля `cert-manager` необходимо перевести режим работы HTTPS некоторых компонентов (например `user-authn` или `prometheus`) на альтернативные варианты работы, либо изменить глобальный параметр, отвечающий за режим работы HTTPS в кластере.

Отобразить список модулей, которые не поддерживаются в Deckhouse CSE и будут отключены, можно следующей командой:

```
echo $MODULES_WILL_DISABLE
```

i Проверьте список и убедитесь, что функциональность указанных модулей не задействована в кластере, и вы готовы к их отключению.

Отключите неподдерживаемые в Deckhouse CSE модули:

```
echo $MODULES_WILL_DISABLE |
tr ' ' '\n' | awk {'print "d8 k -n d8-system exec deploy/deckhouse -- deckhouse-controller
module disable",$1'} | bash
```

В Deckhouse CSE не поддерживается компонент `earlyOOM`. Отключите его с помощью настройки.

Дождитесь перехода пода Deckhouse в статус **Ready** и выполнения всех задач в очереди.

```
d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller queue list
```

Проверьте, что отключенные модули перешли в состояние **Disabled**.

```
d8 k get modules
```

- Создайте ресурс `NodeGroupConfiguration`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: cse-set-sha-images.sh
spec:
  nodeGroups:
    - '*'
  bundles:
    - '*'
  weight: 50
  content: |
    _on_containerd_config_changed() {
      bb-flag-set containerd-need-restart
    }
    bb-event-on 'containerd-config-file-changed' '_on_containerd_config_changed'

    bb-sync-file /etc/containerd/conf.d/cse-sandbox.toml - containerd-config-file-changed <<
EOF_TOML
[plugins]
[plugins."io.containerd.grpc.v1.cri"]
  sandbox_image = "registry-cse.deckhouse.ru/deckhouse/cse@$CSE_SANDBOX_IMAGE"
EOF_TOML

sed -i 's|image: .*|image: registry-cse.deckhouse.ru/deckhouse/cse@$CSE_K8S_API_PROXY|'
/var/lib/bashible/bundle_steps/051_pull_and_configure_kubernetes_api_proxy.sh
sed -i 's|crictl pull .*|crictl pull registry-
cse.deckhouse.ru/deckhouse/cse@$CSE_K8S_API_PROXY|'
/var/lib/bashible/bundle_steps/051_pull_and_configure_kubernetes_api_proxy.sh
EOF
```

Дождитесь завершения синхронизации bashible на всех узлах.

Состояние синхронизации можно отследить по значению `UPTODATE` статуса (отображаемое число узлов в этом статусе должно совпадать с общим числом узлов (`NODES`) в группе):

```
d8 k get ng -O custom-columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate -W
```

В журнале systemd-сервиса bashible на узлах должно появиться сообщение `Configuration is in sync, nothing to do` в результате выполнения следующей команды:

```
journalctl -u bashible -n 5
```

Пример вывода:

```
Aug 21 11:04:28 master-ee-to-cse-0 bashible.sh[53407]: Configuration is in sync, nothing to do.
Aug 21 11:04:28 master-ee-to-cse-0 bashible.sh[53407]: Annotate node master-ee-to-cse-0 with
annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-cse-0 bashible.sh[53407]: Successful annotate node master-ee-to-
cse-0 with annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-cse-0 systemd[1]: bashible.service: Deactivated successfully.
```

7. Актуализируйте секрет доступа к registry Deckhouse CSE, выполнив следующую команду:

```
d8 k -n d8-system create secret generic deckhouse-registry \
  --from-literal=".dockerconfigjson"="{\"auths\": { \"registry-cse.deckhouse.ru\": {
  \"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\": \"${AUTH_STRING}\
  }}}" \
  --from-literal="address"=registry-cse.deckhouse.ru \
  --from-literal="path"=/deckhouse/cse \
  --from-literal="scheme"=https \
  --type=kubernetes.io/dockerconfigjson \
  --dry-run='client' \
  -o yaml | d8 k -n d8-system exec -i svc/deckhouse-leader -C deckhouse -- d8 k replace -f -
```

8. Измените образ Deckhouse на образ Deckhouse CSE:

Команда для Deckhouse CSE версии 1.58:

```
d8 k -n d8-system set image deployment/deckhouse kube-rbac-proxy=registry-
cse.deckhouse.ru/deckhouse/cse@$CSE_DECKHOUSE_KUBE_RBAC_PROXY deckhouse=registry-
cse.deckhouse.ru/deckhouse/cse:$DECKHOUSE_VERSION
```

Команда для Deckhouse CSE версии 1.64 и 1.67:

```
d8 k -n d8-system set image deployment/deckhouse init-downloaded-modules=registry-
cse.deckhouse.ru/deckhouse/cse@$CSE_DECKHOUSE_INIT_CONTAINER kube-rbac-proxy=registry-
cse.deckhouse.ru/deckhouse/cse@$CSE_DECKHOUSE_KUBE_RBAC_PROXY deckhouse=registry-
cse.deckhouse.ru/deckhouse/cse:$DECKHOUSE_VERSION
```

9. Дождитесь перехода пода Deckhouse в статус `Ready` и выполнения всех задач в очереди. Если в процессе возникает ошибка `ImagePullBackOff`, подождите автоматического перезапуска пода.

Чтобы узнать статус пода Deckhouse, используйте следующую команду:

```
d8 k -n d8-system get po -l app=deckhouse
```

Чтобы проверить состояние очереди Deckhouse, используйте следующую команду:

```
d8 k -n d8-system exec deploy/deckhouse -C deckhouse -- deckhouse-controller queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

10. Проверьте, не осталось ли в кластере подов с адресом registry для Deckhouse EE:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[]
| select(.image | contains("deckhouse.ru/deckhouse/ee"))) | .metadata.namespace + "\t" +
.metadata.name' | sort | uniq
```

Если в выводе присутствуют поды модуля `chrony`, заново включите данный модуль (в Deckhouse CSE этот модуль по умолчанию выключен):

```
d8 k -n d8-system exec deploy/deckhouse -- deckhouse-controller module enable chrony
```

11. Очистите временные файлы, ресурс `NodeGroupConfiguration` и переменные:

- Удалите временный файл:

```
rm /tmp/cse-deckhouse-registry.yaml
```

- Удалите ресурс `NodeGroupConfiguration`:

```
d8 k delete ngc containerd-cse-config.sh cse-set-sha-images.sh
```

- Удалите под:

```
d8 k delete pod cse-image
```

- Создайте и примените временный ресурс `NodeGroupConfiguration` для очистки:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: del-temp-config.sh
spec:
  nodeGroups:
    - '*'
  bundles:
    - '*'
  weight: 90
  content: |
    if [ -f /etc/containerd/conf.d/cse-registry.toml ]; then
      rm -f /etc/containerd/conf.d/cse-registry.toml
    fi
    if [ -f /etc/containerd/conf.d/cse-sandbox.toml ]; then
      rm -f /etc/containerd/conf.d/cse-sandbox.toml
    fi
EOF
```

После синхронизации (статус синхронизации на узлах можно отследить по значению `UPTODATE` у NodeGroup) удалите созданный ресурс NodeGroupConfiguration:

```
d8 k delete ngc del-temp-config.sh
```

Переключение DKP на CE/BE/SE/SE+/EE

Переключение DKP на CE/BE/SE/SE+/EE может быть выполнено одним из следующих способов:

- с помощью модуля `registry`,
- без использования модуля `registry`.



- Работоспособность инструкции подтверждена только для версий Deckhouse от `v1.70`. Если ваша версия младше, используйте соответствующую ей документацию.
- Для коммерческих изданий требуется действующий лицензионный ключ с поддержкой нужного издания. При необходимости можно запросить временный ключ.
- Инструкция подразумевает использование публичного адреса container registry: `registry.deckhouse.ru`. В случае использования другого адреса container registry измените команды или воспользуйтесь инструкцией по переключению Deckhouse на использование стороннего registry.
- В редакциях Deckhouse CE/BE/SE/SE+ не поддерживается работа облачных провайдеров Basis Dynamix, Openstack, VCD, vSphere (vSphere поддерживается в редакции SE+) и ряда модулей.
- Все команды выполняются на master-узле существующего кластера под пользователем `root`.
- Если в процессе переключения образ какого-либо модуля не загрузился заново и не модуль переустановился, для устранения проблемы воспользуйтесь инструкцией.

Нижe описаны шаги для переключения кластера с любой редакцией на одну из поддерживаемых: Community Edition, Basic Edition, Standard Edition, Standard Edition+, Enterprise Edition.

Переключение с помощью модуля registry

1. Убедитесь, что кластер был переключен на использование модуля `registry`. Если модуль не используется, перейдите к инструкции.
2. Подготовьте переменные с лицензионным ключом и названием новой редакции:



Заполнять переменную `LICENSE_TOKEN` при переключении на редакцию CE не требуется. Значение переменной `NEW_EDITION` должно быть равно желаемой редакции DKP, например для переключения на редакцию:

- CE, переменная должна быть `ce`;
- BE, переменная должна быть `be`;
- SE, переменная должна быть `se`;
- SE+, переменная должна быть `se-plus`;
- EE, переменная должна быть `ee`.

```
NEW_EDITION=<PUT_YOUR_EDITION_HERE>
LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
```

3. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller queue list
```

Пример вывода (очереди пусты):

Summary:

- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.

4. Запустите временный под Deckhouse новой редакции, чтобы получить актуальные дайджесты и список модулей:

Для CE редакции:

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
$NF}')
d8 k run $NEW_EDITION-image --
image=registry.deckhouse.ru/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION --command sleep -- infinity
```

Для других редакций:

```
d8 k create secret docker-registry $NEW_EDITION-image-pull-secret \
--docker-server=registry.deckhouse.ru \
--docker-username=license-token \
--docker-password=${LICENSE_TOKEN}

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
$NF}')
d8 k run $NEW_EDITION-image \
--image=registry.deckhouse.ru/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION \
--overrides="{\"spec\": {\"imagePullSecrets\": [{\"name\": \"$NEW_EDITION-image-pull-
secret\"}]}}\" \
--command sleep -- infinity
```

Как только под перейдёт в статус **Running**, выполните следующие команды:

```
NEW_EDITION_MODULES=$(d8 k exec $NEW_EDITION-image -- ls -l deckhouse/modules/ | grep -oE "\d.*-\w*" |
awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -F xv -f <(echo $NEW_EDITION_MODULES
| tr ' ' '\n'))
```

5. Убедитесь, что используемые в кластере модули поддерживаются в желаемой редакции.

Посмотреть список модулей, которые не поддерживаются в новой редакции и будут отключены, можно с помощью команды:

```
echo $MODULES_WILL_DISABLE
```

i Проверьте полученный список и убедитесь, что функциональность указанных модулей не используется вами в кластере и вы готовы их отключить.

Отключите неподдерживаемые новой редакцией модули:

```
echo $MODULES_WILL_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module disable",$1'} | bash
```

Дождитесь, пока под Deckhouse перейдёт в состояние **Ready** и убедитесь в выполнении всех задач в очереди:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller queue list
```

Пример вывода (очереди пусты):

Summary:

- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.

6. Удалите созданный секрет и под:

```
d8 k delete pod/$NEW_EDITION-image
d8 k delete secret/$NEW_EDITION-image-pull-secret
```

7. Выполните переключение на новую редакцию. Для этого укажите следующие параметры в ModuleConfig `deckhouse` (для подробной настройки ознакомьтесь с конфигурацией модуля `deckhouse`):

```
---
# Пример для Direct режима
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        # Relax mode используется для проверки наличия текущей версии Deckhouse в указанном
        registry
        # Для переключения между редакциями необходимо использовать данный режим проверки
        registry
        checkMode: Relax
        # Укажите свой параметр <NEW_EDITION>
        imagesRepo: registry.deckhouse.ru/deckhouse/<NEW_EDITION>
        scheme: HTTPS
        # Укажите свой параметр <LICENSE_TOKEN>
        # Если переключение выполняется на CE редакцию, удалите данный параметр
        license: <LICENSE_TOKEN>
---
# Пример для Unmanaged режима
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        # Relax mode используется для проверки наличия текущей версии Deckhouse в указанном
        registry
        # Для переключения между редакциями необходимо использовать данный режим проверки
        checkMode: Relax
        # Укажите свой параметр <NEW_EDITION>
        imagesRepo: registry.deckhouse.ru/deckhouse/<NEW_EDITION>
        scheme: HTTPS
        # Укажите свой параметр <LICENSE_TOKEN>
        # Если переключение выполняется на CE редакцию, удалите данный параметр
        license: <LICENSE_TOKEN>
```

8. Дождитесь переключения `registry`. Для проверки выполнения переключения воспользуйтесь инструкцией.

Пример вывода:

```
conditions:
- lastTransitionTime: "...
  message: |-
    Mode: Relax
    registry.deckhouse.ru: all 1 items are checked
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
```

9. После переключения, удалите из ModuleConfig `deckhouse` параметр `checkMode: Relax`, чтобы активировать выполнение проверки по умолчанию. Удаление запустит проверку наличия критически важных компонентов в registry.
10. Дождитесь выполнения проверки. Статус переключения режима registry можно получить, воспользовавшись инструкцией.

Пример вывода:

```
conditions:
- lastTransitionTime: "...
  message: |-
    Mode: Default
    registry.deckhouse.ru: all 155 items are checked
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
```

11. Проверьте, нет ли в неймспейсах `d8-*` подов в состоянии ошибки, которые не могут загрузить образы. Это необходимо сделать вручную, так как в настоящий момент модули Deckhouse не переинициализируются автоматически после изменений, описанных выше.

Получите список подов:

```
d8 k get po -A
```

Получите детальную информацию о проблемных подах:

```
d8 k describe po <pod_name> <namespace>
```

Повторно загрузите соответствующие проблемным подам модули, выполнив на всех master-узлах команду:

```
rm -rf /var/lib/deckhouse/downloaded/<module-name>/
```

Для получения `<module-name>` выполните команду:

```
d8 k get modules
```

После удаления данных нужных модулей перезапустите Deckhouse:

```
d8 k rollout restart deploy -n d8-system deckhouse
```

12. Проверьте, не осталось ли в кластере подов со старым адресом registry, где `<YOUR-PREVIOUS-EDITION>` — название вашей прошлой редакции:

Для Unmanaged-режима:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] | select(.image | contains("deckhouse.ru/deckhouse/<YOUR-PREVIOUS-EDITION>")) | .metadata.namespace + "\t" + .metadata.name' | sort | uniq
```

Для других режимов, использующих фиксированный адрес (данная проверка не учитывает внешние модули):


```
# Получаем список актуальных digest'ов из файла images_digests.json внутри Deckhouse
IMAGES_DIGESTS=$(d8 k -n d8-system exec -i svc/deckhouse-leader -C deckhouse -- cat
/deckhouse/modules/images_digests.json | jq -r '[][]' | sort -u)

# Проверяем, есть ли поды, использующие образы Deckhouse по адресу `registry.d8-
system.svc:5001/system/deckhouse`
# с digest'ом, отсутствующим в списке актуальных digest'ов из IMAGES_DIGESTS
d8 k get pods -A -O json |
jq -r --argjson digests "$((printf '%s\n' $IMAGES_DIGESTS | jq -R . | jq -s .))" '
.items[]
| {name: .metadata.name, namespace: .metadata.namespace, containers: .spec.containers}
| select(.containers != null)
| select(
  .containers[]
  | select(.image | test("registry.d8-system.svc:5001/system/deckhouse") and
test("@sha256:"))
  | .image as $img
  | ($img | split("@") | last) as $digest
  | ($digest | IN($digests[]) | not)
)
| .namespace + "\t" + .name
' | sort -u
```

Переключение без использования модуля registry

1. Если модуль `registry` включен, отключите его с помощью инструкции.
2. Подготовьте переменные с лицензионным ключом и названием новой редакции:

i Заполнять переменные `NEW_EDITION` и `AUTH_STRING` при переключении на редакцию CE не требуется. Значение переменной `NEW_EDITION` должно быть равно желаемой редакции DKP, например для переключения на редакцию:

- CE, переменная должна быть `ce`;
- BE, переменная должна быть `be`;
- SE, переменная должна быть `se`;
- SE+, переменная должна быть `se-plus`;
- EE, переменная должна быть `ee`.

```
NEW_EDITION=<PUT_YOUR_EDITION_HERE>
LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64)"
```

3. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

4. Создайте ресурс `NodeGroupConfiguration` для переходной авторизации в `registry.deckhouse.ru`:

i Перед созданием ресурса ознакомьтесь с разделом [Как добавить конфигурацию для дополнительного registry](#).
При переходе на редакцию Deckhouse CE пропустите этот шаг.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-$NEW_EDITION-config.sh
spec:
  nodeGroups:
    - '*'
  bundles:
    - '*'
  weight: 30
  content: |
    _on_containerd_config_changed() {
      bb-flag-set containerd-need-restart
    }
    bb-event-on 'containerd-config-file-changed' '_on_containerd_config_changed'
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/$NEW_EDITION-registry.toml - containerd-config-file-
changed << "EOF_TOML"
    [plugins]
    [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry.configs]
    [plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.ru".auth]
      auth = "$AUTH_STRING"
    EOF_TOML
EOF
```

Дождитесь появления файла `/etc/containerd/conf.d/$NEW_EDITION-registry.toml` на узлах и завершения синхронизации `bashible`. Чтобы отследить статус синхронизации, проверьте значение `UPTODATE` (число узлов в этом статусе должно совпадать с общим числом узлов (`NODES`) в группе):

```
d8 k get ng -O custom-columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate -W
```

Пример вывода:

NAME	NODES	READY	UPTODATE
master	1	1	1
worker	2	2	2

Также в журнале `systemd`-сервиса `bashible` должно появиться сообщение `Configuration is in sync, nothing to do` в результате выполнения следующей команды:

```
journalctl -u bashible -n 5
```

Пример вывода:

```
Aug 21 11:04:28 master-ee-to-se-0 bashible.sh[53407]: Configuration is in sync, nothing to do.
Aug 21 11:04:28 master-ee-to-se-0 bashible.sh[53407]: Annotate node master-ee-to-se-0 with
annotation node.deckhouse.io/ configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-se-0 bashible.sh[53407]: Successful annotate node master-ee-to-se-
0 with annotation node.deckhouse.io/ configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-se-0 systemd[1]: bashible.service: Deactivated successfully.
```

- Запустите временный под Deckhouse новой редакции, чтобы получить актуальные дайджесты и список модулей:

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
$NF}')
d8 k run $NEW_EDITION-image --
image=registry.deckhouse.ru/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION --command sleep -- infinity
```

- После перехода пода в статус `Running` выполните следующие команды:

```
NEW_EDITION_MODULES=$(d8 k exec $NEW_EDITION-image -- ls -l deckhouse/modules/ | grep -oE "\d.*-\w*" |
awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -Fvx -f <(echo $NEW_EDITION_MODULES
| tr ' ' '\n'))
```

7. Убедитесь, что используемые в кластере модули поддерживаются в желаемой редакции.

Посмотреть список модулей, которые не поддерживаются в новой редакции и будут отключены, можно с помощью команды:

```
echo $MODULES_WILL_DISABLE
```

i Проверьте полученный список и убедитесь, что функциональность указанных модулей не используется в кластере и вы готовы их отключить.

Отключите неподдерживаемые новой редакцией модули:

```
echo $MODULES_WILL_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module disable",$1'} | bash
```

Дождитесь, пока под Deckhouse перейдёт в состояние `Ready` и убедитесь в выполнении всех задач в очереди:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

8. Выполните команду `deckhouse-controller helper change-registry` из пода Deckhouse с параметрами новой редакции:

Для переключения на BE/SE/SE+/EE издания:

```
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.ru\": {\"username\": \"license-
token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\": \"${AUTH_STRING}\"}}\" | base64 -w 0)
d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry --type merge --patch="
{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller helper change-registry --
user=license-token --password=$LICENSE_TOKEN --new-deckhouse-tag=$DECKHOUSE_VERSION
registry.deckhouse.ru/deckhouse/$NEW_EDITION
```

Для переключения на CE издание:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller helper change-registry --new-
deckhouse-tag=$DECKHOUSE_VERSION registry.deckhouse.ru/deckhouse/ce
```

9. Проверьте, не осталось ли в кластере подов со старым адресом `registry`, где `<YOUR-PREVIOUS-EDITION>` — название вашей прошлой редакции:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] | select(.image |
contains("deckhouse.ru/deckhouse/<YOUR-PREVIOUS-EDITION>")) | .metadata.namespace + "\t" +
.metadata.name' | sort | uniq
```

10. Удалите временные файлы, ресурс `NodeGroupConfiguration` и переменные:

i При переходе на редакцию CE пропустите этот шаг.

```
d8 k delete ngc containerd-$NEW_EDITION-config.sh
d8 k delete pod $NEW_EDITION-image
d8 k apply -f - <<EOF
  apiVersion: deckhouse.io/v1alpha1
  kind: NodeGroupConfiguration
  metadata:
    name: del-temp-config.sh
  spec:
    nodeGroups:
      - '*'
    bundles:
      - '*'
    weight: 90
    content: |
      if [ -f /etc/containerd/conf.d/$NEW_EDITION-registry.toml ]; then
        rm -f /etc/containerd/conf.d/$NEW_EDITION-registry.toml
      fi
EOF
```

После завершения синхронизации bashible (статус синхронизации на узлах отображается по значению `UPTODATE` у NodeGroup) удалите созданный ресурс NodeGroupConfiguration:

```
d8 k delete ngc del-temp-config.sh
```

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `deckhouse`
- Документация модуля `node-manager`
- Документация модуля `registry`

Аутентификация

Аутентификация — это процесс проверки подлинности пользователя, обеспечивающий контроль доступа ко всем интерфейсам Deckhouse Kubernetes Platform (DKP) и ресурсам кластера. Платформа реализует сквозную аутентификацию, позволяя применять единый механизм как для собственных компонентов, так и для пользовательских приложений.

В основе механизма лежит федеративный OpenID Connect (OIDC) провайдер — `Dex`. Подробнее об устройстве аутентификации можно узнать в разделе Архитектура.

В зависимости от конфигурации, DKP поддерживает два подхода к аутентификации:

- Локальная аутентификация — пользователи и группы создаются непосредственно в кластере, данные хранятся в специальных ресурсах `User` и `Group`. При этом в ресурсе `User` сохраняется не пароль в открытом виде, а его зашифрованная хеш-сумма (`bcrypt`).
- Интеграция с внешними провайдерами — используется подключение к системам, таким как LDAP, GitLab, GitHub и другим, для реализации единой учётной записи и аутентификации в нескольких кластерах DKP.

С точки зрения пользователя кластера или разработчика приложения, не имеет значения то, как администратор настроил аутентификацию в DKP — интерфейс аутентификации для пользователя и способы включения аутентификации для приложения будут одинаковыми. О настройке аутентификации для приложений можно узнать в разделе Использование → IAM → Аутентификация.

Платформа также предоставляет возможности:

- Включение аутентификации в любом веб-приложении кластера.
- Организация доступа с аутентификацией к Kubernetes API через балансировщик трафика.

Дополнительные ресурсы

- Кастомные ресурсы модуля `user-authn`

Интеграция с внешними провайдерами аутентификации

Подключение внешнего провайдера аутентификации позволяет использовать единые учетные данные для входа в несколько кластеров и одновременно работать с несколькими провайдерами.

DKP поддерживает подключение следующих внешних провайдеров и протоколов аутентификации:

- LDAP (например, Active Directory);
- OIDC (например, Okta, Keycloak, Gluu, Blitz Identity Provider);
- GitHub;
- GitLab;
- Atlassian Crowd;
- Bitbucket Cloud.

- i** Политика безопасности паролей (требования к сложности, срок действия, история, двухфакторная аутентификация и т.д.) полностью контролируется внешним провайдером аутентификации. Deckhouse не управляет паролями и не вмешивается в реализацию этих политик на стороне провайдера.

Общая схема интеграции

- i** Параметр `allowedGroups` в ресурсе `DexProvider` позволяет ограничить вход только пользователями, входящим в указанные группы. Если список `allowedGroups` задан, пользователь обязан состоять хотя бы в одной из этих групп — иначе аутентификация будет считаться неудачной. Если параметр не указан, фильтрация по группам не применяется.

1. Создайте OAuth-приложение у провайдера аутентификации:
 - о укажите Redirect URI вида `https://dex.<publicDomainTemplate>/callback`;
 - о получите `clientId` и `clientSecret`.

- i** **Важно.** При указании Redirect URI подставьте значение `publicDomainTemplate` без `%s`. Например, если указано `publicDomainTemplate: '%s.sandbox1.deckhouse-docs.flant.com'`, то фактический URI будет `https://dex.sandbox20.deckhouse-docs.flant.com/callback`.

Для того, чтобы узнать адрес Dex (URI), выполните команду:

```
d8 k -n d8-user-authn get ingress dex -o jsonpath="{.spec.rules[*].host}"
```

2. Создайте ресурс `DexProvider` с учётом специфики выбранного провайдера.
3. Включите модуль `user-authn` (если он выключен).

Сделать это можно как через веб-интерфейс администратора, так и через CLI. Далее приведен пример работы через Deckhouse CLI (требуется настроенный на работу с кластером контекст `kubectl`).

Проверьте статус модуля:

```
d8 k get module user-authn
```

Пример вывода:

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
user-authn		Embedded	Available	True	True

Включите модуль через CLI:

```
d8 system module enable user-authn
```

4. Настройте модуль `user-authn`.
 - о Откройте настройки модуля `user-authn` (создайте ресурс `ModuleConfig user-authn`, если его нет):

```
d8 k edit mc user-authn
```

- о Укажите необходимые параметры модуля в секции `spec.settings`. Подробнее о настройках модуля `user-authn` можно узнать в разделе справки модуля.

Пример конфигурации `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    kubeconfigGenerator:
      - id: direct
        masterURI: https://159.89.5.247:6443
        description: "Direct access to kubernetes API"
    publishAPI:
      enabled: true
```

После настройки интеграции в кластере с внешним провайдером аутентификации, аутентификация через него станет возможна в веб-интерфейсах платформы. О настройке аутентификации для пользовательских приложений можно узнать в разделе [Использование → IAM → Аутентификация](#).

Интеграция по OIDC (OpenID Connect)

Аутентификация через OIDC-провайдера требует регистрации клиента (или создания приложения). Сделайте это по документации вашего провайдера (например, Okta, Keycloak, Gluu или Blitz).

Полученные в ходе выполнения инструкции `clientId` и `clientSecret` укажите в ресурсе `DexProvider`.

i При регистрации приложения в любом OIDC-провайдере необходимо указать адрес перенаправления (Redirect URI). Для интеграции с `DexProvider` используйте следующий формат: `https://dex.<publicDomainTemplate>/callback`, где `publicDomainTemplate` — шаблон DNS-имен вашего кластера, определенный в модуле `global`.

i Для обеспечения корректного отзыва токена при выходе из приложения и требования повторной авторизации установите параметр `prompt` в значение `login`. Это гарантирует, что пользователю будет предложено повторно ввести учетные данные при повторной аутентификации.

Чтобы настроить детальный (гранулированный) доступ пользователей к приложениям:

- добавьте параметр `allowedUserGroups` в `ModuleConfig` нужного приложения;
- добавьте соответствующие группы пользователю, используя те же наименования групп как на стороне провайдера, так и на стороне Deckhouse.

Keycloak

В процессе настройки Keycloak выберите подходящий `realm`, добавьте пользователя в `Users` и создайте клиент в разделе `Clients` с включённой аутентификацией, необходимой для генерации `clientSecret`. Затем выполните следующие шаги:

1. Создайте в разделе `Client scopes` `scope` с именем `groups`, и назначьте ему предопределенный маппинг `Group Membership` («Client scopes» → «Client scope details» → «Mappers» → «Configure a new mapper»). В поле «Name» и «Token Claim Name» впишите `groups`, в параметре «Full group path» задайте `off`.
2. В созданном ранее клиенте добавьте данный `scope` во вкладке `Client scopes` («Clients» → «Client details» → «Client Scopes» → «Add client scope»).
3. В полях «Valid redirect URIs», «Valid post logout redirect URIs» и «Web origins» конфигурации клиента укажите `https://dex.<publicDomainTemplate>/*`, где `publicDomainTemplate` — это указанный шаблон DNS-имен кластера в модуле `global`.

Пример настройки провайдера для интеграции с Keycloak:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Используйте имя вашего realm
    clientId: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups
```

Если в Keycloak не используется подтверждение учетных записей по email, для корректной работы с ним в качестве провайдера аутентификации внесите изменения в настройку `Client scopes` одним из следующих способов:

- Удалите маппинг `Email verified` («Client Scopes» → «Email» → «Mappers»). Это необходимо для корректной обработки значения `true` в поле `insecureSkipEmailVerified` и правильной выдачи прав пользователям с неподтвержденным email.
- Если отредактировать или удалить маппинг `Email verified` невозможно, создайте отдельный Client Scope с именем `email_dkp` (или любым другим) и добавьте в него два маппинга:
 - `email`: «Client Scopes» → `email_dkp` → «Add mapper» → «From predefined mappers» → `email`;
 - `email verified`: «Client Scopes» → `email_dkp` → «Add mapper» → «By configuration» → «Hardcoded claim». Укажите следующие поля:
 - «Name»: `email verified`;
 - «Token Claim Name»: `emailVerified`;
 - «Claim value»: `true`;
 - «Claim JSON Type»: `boolean`.

После этого в клиенте, зарегистрированном для кластера DKP, в разделе «Clients» для `Client scopes` замените значение `email` на `email_dkp`.

В ресурсе DexProvider укажите параметр `insecureSkipEmailVerified: true` и в поле `.spec.oidc.scopes` замените название Client Scope на `email_dkp`, следуя примеру:

```
scopes:
- openid
- profile
- email_dkp
- groups
```

Blitz Identity Provider

Пример настройки провайдера для интеграции с Blitz Identity Provider:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim для получения групп пользователя, группы пользователя настраиваются
на стороне провайдера Blitz Identity Provider.
    clientID: clientID
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipEmailVerified: true # Установить true, если нет необходимости в проверке email
пользователя.
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
      - profile
      - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC
```

Okta

Пример настройки провайдера для интеграции с Okta:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
```

После включения интеграции с Okta можно использовать группы пользователей из Okta, для управления правами. Например, можно задать список групп, пользователи из которых получают доступ к Grafana.

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access

```

Интеграция по SAML 2.0

В примере представлены настройки провайдера для интеграции с SAML 2.0 Identity Provider (например, AD FS, Okta, Keycloak).

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Корпоративный SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent

```

Для настройки SAML Identity Provider:

1. Зарегистрируйте Dex как Service Provider (SP) в вашем IdP со следующими параметрами:
 - **ACS URL (Assertion Consumer Service):** `https://dex.<modules.publicDomainTemplate>/callback`;
 - **Entity ID:** `https://dex.<modules.publicDomainTemplate>/callback`;
 - **Формат NameID:** `persistent` или `emailAddress`.
2. Настройте маппинг атрибутов в IdP для отправки атрибутов `email`, `name` (имя пользователя) и `groups` в SAML assertion.
3. Экспортируйте сертификат подписи IdP и укажите его в поле `rootCAData` ресурса `DexProvider`.

i SAML не поддерживает refresh tokens нативно. Dex кеширует identity пользователя из первичного SAML assertion и возвращает её при последующих запросах refresh. Время жизни сессии контролируется настройками `expiry.refreshTokens` в конфигурации модуля `user-authn`.

Интеграция по LDAP

Для настройки аутентификации создайте в LDAP учетную запись с правами только на чтение (service account). Эта учетная запись будет использоваться для выполнения поисковых запросов в каталоге LDAP.

В ресурсе `DexProvider` укажите следующие параметры:

- `bindDN`: Полный DN (Distinguished Name) созданного service account. Например: `cn=readonly,dc=example,dc=org`.
- `bindPW`: Пароль для указанного `bindDN`.

i Если ваш LDAP-сервер позволяет анонимный доступ для выполнения поисковых запросов, параметры `bindDN` и `bindPW` можно опустить. Однако, рекомендуется использовать аутентифицированный доступ для повышения безопасности.

В параметре `bindPW` укажите пароль в открытом виде (plain text). Dex не поддерживает передачу хешированных паролей в этом параметре.

Также вы можете включить поддержку:

- Базовой аутентификации (Basic Authentication) для доступа к Kubernetes API, используя учетные данные LDAP. Для этого установите параметр `enableBasicAuth: true` в ресурсе `DexProvider`. Подробнее в разделе Доступ с

использованием Basic Authentication.

- SSO по Kerberos (SPNEGO) для LDAP. Для этого в ресурсе DexProvider включите блок `spec.ldap.kerberos` и настройте в нём необходимые параметры. Подробнее в разделе SSO по Kerberos (SPNEGO) для LDAP.

Пример настройки провайдера для интеграции с Active Directory (базовая аутентификация):

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    insecureSkipVerify: true

    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!

    enableBasicAuth: true

    usernamePrompt: Email Address

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
        - userAttr: DN
          groupAttr: member
          nameAttr: cn
```

Интеграция с GitHub

В организации GitHub необходимо создать новое приложение.

Для этого выполните следующие шаги:

1. Перейдите в «Settings» → «Developer settings» → «OAuth App» → «New OAuth App», и в качестве «Authorization callback URL» укажите адрес `https://dex.<publicDomainTemplate>/callback`.
2. Полученные `Client ID` и `Client Secret` укажите в ресурсе DexProvider.

Если организация GitHub находится под управлением клиента:

1. Перейдите в «Settings» → «Applications» → «Authorized OAuth Apps» → `<имя созданного OAuth-приложения>` и нажмите «Send Request» для подтверждения.
2. Попросите клиента подтвердить запрос, который придет к нему на email.

Пример настройки провайдера для интеграции с GitHub:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company GitHub
  # Опционально: временно отключить провайдер, не удаляя CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

Интеграция с GitLab

В GitLab проекта необходимо создать новое приложение.

Для этого выполните следующие шаги:

1. Self-hosted-версия GitLab: перейдите в «Admin Area» → «Applications» → «New application» и в качестве

- «Redirect URI (Callback URL)» укажите адрес `https://dex.<publicDomainTemplate>/callback`, а также выберите «scopes»: `read_user`, `openid`.
- 2. GitLab Cloud (gitlab.com): под главной учетной записью проекта перейдите в «User Settings» → «Applications» → «Add new application» и в качестве «Redirect URI (Callback URL)» укажите адрес `https://dex.<publicDomainTemplate>/callback`, а также выберите «scopes»: `read_user`, `openid`.
- 3. Полученные `Application ID` и секрет укажите в ресурсе `DexProvider`.



Для GitLab версии 16 и выше включите опцию «Trusted» при создании приложения. Эта опция доступна при создании приложений в «Admin Area» → «Applications». Установка приложения как доверенного позволяет пропустить шаг авторизации для пользователей, что может быть полезно в контролируемых средах.

Пример настройки провайдера для интеграции с GitLab:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated Gitlab
  gitlab:
    baseUrl: https://gitlab.example.com
    clientId: plainstring
    clientSecret: plainstring
    groups:
      - administrators
      - users
```

Интеграция с Atlassian Crowd

В соответствующем проекте Atlassian Crowd необходимо создать новое Generic-приложение.

Для этого выполните следующие шаги:

1. Перейдите в «Applications» → «Add application».
2. Полученные «Application Name» и «Password» укажите в ресурсе `DexProvider`.
3. При указании групп в ресурсе `DexProvider` убедитесь, что их названия приведены к нижнему регистру (lowercase). Это необходимо для корректного сопоставления групп между Crowd и Deckhouse.

Пример настройки провайдера для интеграции с Atlassian Crowd:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseUrl: https://crowd.example.com/crowd
    clientId: plainstring
    clientSecret: plainstring
    enableBasicAuth: true
    groups:
      - administrators
      - users
```

Интеграция с Bitbucket Cloud

Для настройки аутентификации необходимо в меню команды Bitbucket создать нового OAuth пользователя (consumer).

Для этого выполните следующие шаги:

1. Перейдите в «Personal settings» → «Access management» → «OAuth consumers» → «Add consumer» и в качестве «Callback URL» укажите адрес `https://dex.<publicDomainTemplate>/callback`.
2. Разрешите доступ: «Account: Read» → позволяет получать основную информацию о пользователе (например, email, имя пользователя), «Workspace membership → Read»: позволяет получать информацию о членстве пользователя в рабочих пространствах.
3. Полученные `Key` и секрет укажите в ресурсе `DexProvider`.

Пример настройки провайдера для интеграции с Bitbucket:

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
  teams:
    - administrators
    - users

```

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `user-authn`

Локальная аутентификация

Помимо внешних провайдеров аутентификации, DKP позволяет использовать локальную аутентификацию.

Локальная аутентификация обеспечивает проверку и управление доступом пользователей с возможностью настройки парольной политики, поддержкой двухфакторной аутентификации (2FA) и управлением группами. Реализация соответствует требованиям безопасности ФСТЭК и рекомендациям OWASP, обеспечивая надёжную защиту доступа к кластеру и приложениям без необходимости интеграции с внешними системами аутентификации.

Локальная аутентификация подразумевает создание в кластере объектов User и Group для статических пользователей и групп:

- В объекте User хранится информация о пользователе, включая email и хеш пароля (пароль в явном виде не сохраняется).
- В объекте Group задаётся список пользователей, объединённых в группу.

Создание статического пользователя

Для создания статического пользователя создайте ресурс User.

Пример создания ресурса (обратите внимание, что в приведенном примере указан `ttl`):

```

apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  password: $2a$10$etblbZ9yfZaKgbvysf1qguW3WULdMnxwWFrkoKpRH1yewa5etjjAa
  ttl: 24h

```

Придумайте пароль и укажите в поле `password` его bcrypt-хеш в исходном виде или закодированном в Base64. Чтобы получить bcrypt-хеш в Base64, выполните команду:

```
echo -n '<PASSWORD>' | httpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo
```

i

Если команда `httpasswd` недоступна, установите соответствующий пакет в зависимости от дистрибутива:

- `apache2-utils` — для дистрибутивов, основанных на Debian;
- `httpd-tools` — для дистрибутивов, основанных на CentOS;
- `apache2-httpasswd` — для ALT Linux.

Операции над локальным пользователем

Операции сброса пароля, сброса 2FA и блокировки пользователя выполняются через ресурс `UserOperation`. В поле `initiatorType` указывается, кто инициировал операцию: администратор (`admin`), система (`system`) или сам пользователь (`self`).

Административные операции

Для административных действий над локальными пользователями используйте команды `d8 iam user`. Они создают ресурс `UserOperation` с `initiatorType: admin`, дожидаются выполнения операции и выводят результат.

При выполнении операций `ResetPassword`, `Reset2FA` и `Lock` удаляются объекты `Dex OfflineSessions` и `RefreshToken`, принадлежащие пользователю. Это завершает активные offline-сессии пользователя и требует

повторной аутентификации.

Примеры использования команд `d8 iam user`:

- интерактивный сброс пароля:

```
d8 iam user reset-password admin
```

- сброс пароля с чтением нового из stdin:

```
echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
```

- сброс пароля с автоматической генерацией нового:

```
d8 iam user reset-password admin --generate-password
```

- сброс пароля в захешированном виде (если пароль захеширован, передайте bcrypt-хеш без кодирования в Base64):

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

- сброс 2FA:

```
d8 iam user reset2fa admin
```

- блокировка пользователя на 30 минут:

```
d8 iam user lock admin 30m
```

- разблокировка пользователя:

```
d8 iam user unlock admin
```

По умолчанию команды ожидают завершения операции. Чтобы только создать `UserOperation` и вывести его имя, используйте флаг `--wait=false`.

Сброс пароля пользователем

Локальный пользователь может самостоятельно сбросить свой пароль в интерфейсе аутентификации DKP. При этом создаётся ресурс `UserOperation` с типом `ResetPassword` и `initiatorType: self`.

Самостоятельный сброс пароля доступен только для локальных учётных записей (встроенный коннектор `Local`). Пользователи, которые входят через внешние провайдеры аутентификации, должны обращаться к администратору соответствующей системы.

При сбросе пароля:

- новый пароль должен соответствовать парольной политике;
- завершаются активные сессии пользователя — требуется повторная аутентификация.

Подробнее о сценариях смены и сброса пароля с точки зрения пользователя — в разделе [Настройка аутентификации для приложений](#).

Добавление пользователя в группу



Запрещено использовать пользователей и группы с префиксом `system:`. Аутентификация таких пользователей или участников этих групп будет отклонена, а в логах `kube-apiserver` появится соответствующее предупреждение.

Чтобы объединять статических пользователей в группы, создайте ресурс `Group`.

Пример создания ресурса:

```
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin
```

Здесь `members` — список пользователей, которые входят в группу.

После создания группы и добавления в неё пользователей, необходимо настроить авторизацию.

Настройка парольной политики

Парольная политика позволяет контролировать сложность пароля, ротацию и блокировку пользователей.

Для настройки парольной политики используйте поле `passwordPolicy` в конфигурации модуля `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"
```

Описание полей:

- `complexityLevel` — уровень сложности пароля;
- `passwordHistoryLimit` — число предыдущих паролей, которые хранит система, чтобы предотвратить их повторное использование;
- `lockout` — настройки блокировки при превышении лимита неудачных попыток входа:
 - `lockout.maxAttempts` — лимит неудачных попыток;
 - `lockout.lockDuration` — длительность блокировки пользователя;
- `rotation` — настройки ротации паролей:
 - `rotation.interval` — период обязательной смены пароля.

Настройка двухфакторной аутентификации (2FA)

2FA позволяет повысить уровень безопасности, требуя ввести код из приложения-аутентификатора TOTP (например, Google Authenticator) при входе.

Для настройки 2FA используйте поле `staticUsers2FA` в конфигурации модуля `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    staticUsers2FA:
      enabled: true
      issuerName: "awesome-app"
```

Описание полей:

- `enabled` — включает или отключает 2FA для всех статических пользователей;
- `issuerName` — имя, которое будет отображаться в приложении-аутентификаторе при добавлении аккаунта.

i После включения 2FA каждый пользователь должен пройти процесс регистрации в приложении-аутентификаторе при первом входе.

Дополнительные ресурсы

- Кастомные ресурсы модуля `user-authn`
- Параметры модуля `user-authn`

Доступ к Kubernetes API через балансировщик трафика

С DKP можно использовать аутентификацию при доступе к Kubernetes API. В этом случае, пользователь в веб-интерфейсе kubesonfig DKP может сгенерировать конфигурацию для `kubectl`, для безопасного доступа к Kubernetes API через балансировщик трафика (Ingress-контроллер).

Чтобы настроить доступ, выполните следующие шаги:

1. Включите публикацию Kubernetes API. Для этого установите параметр `publishAPI.enabled: true` в настройках модуля `user-authn` или с помощью веб-интерфейса администратора Deckhouse.

Пример конфигурации модуля:

```
спес:
  enabled: true
  version: 2
  settings:
    publishAPI:
      enabled: true
```

- Откройте веб-интерфейс kubeconfig. Веб-интерфейс для генерации kubeconfig в DKP активируется автоматически после включения параметра `publishAPI` в модуле `user-authn`. Этот веб-интерфейс доступен по URL:

```
https://kubeconfig.<publicDomainTemplate>
```

Например, если `publicDomainTemplate: %s.kube.my`, то URL будет `https://kubeconfig.kube.my`.

- Сгенерируйте конфигурацию `kubectl`. После авторизации в интерфейсе kubeconfig пользователь получит набор команд для настройки `kubectl`. Эти команды можно скопировать и вставить в консоль. Аутентификация будет производиться по OIDC-токену, выданному Dex. При поддержке провайдером функции продления сессии конфигурация будет включать `refresh token`, что позволит продлевать доступ без повторной аутентификации.
- Настройте несколько точек подключения к API. В конфигурации модуля `user-authn` можно задать несколько точек подключения (`kube-apiserver`) с описанием и CA-сертификатами для каждой. Это может понадобиться, если кластер доступен через разные сети — например, VPN или публичный IP:

```
settings:
  kubeconfigGenerator:
    - id: direct
      masterURI: https://159.89.5.247:6443
      description: "Direct access to kubernetes API"
```

Как работает защита доступа к Kubernetes API

В Deckhouse Kubernetes Platform вы можете безопасно опубликовать Kubernetes API наружу с помощью Ingress-контроллера, сохранив контроль над доступом. Публикация API и настройка аутентификации осуществляется через модуль `user-authn`. Вы можете настроить:

- список доверенных IP-адресов или сетей, которым разрешён доступ;
- список групп пользователей, которые имеют право аутентификации;
- Ingress-контроллер, через который будет осуществляться доступ.

Для настройки:

- Включите публикацию API, как в примере выше.
- Настройте ограничения доступа. В конфигурации модуля можно указать:
 - список сетевых адресов, которым разрешён доступ (`allowedSourceRanges`);
 - список групп пользователей, которым разрешено подключение к Kubernetes API (`allowedUserGroups`);
 - выбор Ingress-контроллера, через который будет работать публикация (`ingressClass`).
- Используйте веб-интерфейс kubeconfig. Пользователи смогут получить безопасный доступ к API через kubeconfig, сгенерированный в веб-интерфейсе (`https://kubeconfig.<publicDomainTemplate>`). Этот kubeconfig будет содержать OIDC-токен и настройки подключения через Ingress.

Что будет настроено автоматически при включении публикации API:

- Deckhouse сам настроит необходимые аргументы для `kube-apiserver`;
- будет сгенерирован сертификат CA и добавлен в kubeconfig;
- будет настроен вход через Dex с поддержкой OIDC.

Доступ с использованием базовой аутентификации (LDAP)

Помимо OIDC можно настроить прямой доступ к Kubernetes API с использованием базовой аутентификации (Basic Authentication, по логину и паролю). В этом случае проверка учетных данных осуществляется через LDAP-совместимую службу каталогов.

Для настройки:

- Включите публикацию API (параметр `publishAPI`).
- Настройте провайдер LDAP в модуле `user-authn` и включите в нём опцию `enableBasicAuth: true`.



В кластере может быть только один провайдер с включенным параметром `enableBasicAuth`.

После этого пользователи могут настроить свой kubeconfig, указав логин и пароль LDAP:

```

apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Путь к CA сертификату или insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default

```

SSO по Kerberos (SPNEGO) для LDAP

Dex поддерживает аутентификацию без отображения формы ввода логина/пароля, которая реализуется с помощью механизма Kerberos (SPNEGO) для LDAP-коннектора. Механизм работает по следующему принципу:

1. Браузер, доверяющий хосту Dex, отправляет `Authorization: Negotiate ...`.
2. Dex валидирует Kerberos-билет по keytab и пропускает форму ввода логина/пароля.
3. Dex сопоставляет principal с LDAP-именем, получает группы и завершает OIDC-поток.



Для настройки SSO по Kerberos (SPNEGO) в LDAP должна быть учетная запись с правами только на чтение (service account). Подробнее о настройке — в разделе «Интеграция по LDAP».

Включение SSO по Kerberos (SPNEGO) для LDAP:

1. В инфраструктуре клиента должен быть задан SPN `HTTP/<fqdn-dex>` для сервисного аккаунта и сгенерирован keytab.
2. В кластере создайте секрет в неймспейсе `d8-user-authn` с ключом `krb5.keytab`.
3. В ресурсе DexProvider (тип LDAP) включите блок `spec.ldap.kerberos` и настройте в нём параметры:
 - `enabled: true`;
 - `keytabSecretName: <имя секрета>`;
 - опционально: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`.

Dex автоматически смонтирует keytab и начнёт принимать SPNEGO. `krb5.conf` на сервере не обязателен — билеты проверяются по keytab.

Пример настройки SSO по Kerberos (SPNEGO) для LDAP (расширение спецификации LDAP-провайдера):

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
        - userAttr: uid
          groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab # Секрет в неймспейсе `d8-user-authn` с ключом
      'krb5.keytab'.
      expectedRealm: EXAMPLE.COM # Опционально, проверка realm (без учёта регистра).
      usernameFromPrincipal: sAMAccountName # localpart|sAMAccountName|userPrincipalName
      fallbackToPassword: false # По умолчанию false; если true – при отсутствии/ошибке
заголовка `Authorization: Negotiate` будет показана форма ввода логина/пароля.

```

Примечания:

- Секрет `dex-kerberos-keytab` должен находиться в неймспейсе `d8-user-authn` и содержать ключ `krb5.keytab`.
- Один под Dex может обслуживать несколько LDAP+Kerberos провайдеров. У каждого — свой `keytab`; `krb5.conf` не требуется (Dex проверяет билеты офлайн по `keytab`).

Дополнительные ресурсы

- Документация модуля `user-authn`

Настройка ресурсов

В Deckhouse Kubernetes Platform можно задать значения лимитов запросов и ресурсов для всех компонентов. По умолчанию используются следующие параметры:

- **Dex OIDC провайдер** — 10m CPU / 25Mi памяти (запросы), 100m CPU / 100Mi памяти (лимиты);
- **Генератор kubeconfig** — 10m CPU / 25Mi памяти (запросы), 100m CPU / 100Mi памяти (лимиты);
- **Basic Auth прокси** — 10m CPU / 25Mi памяти (запросы), 100m CPU / 100Mi памяти (лимиты);
- **Dex authenticator** — 10m CPU / 25Mi памяти (запросы), 100m CPU / 100Mi памяти (лимиты);
- **Redis** (используется Dex authenticator) — 10m CPU / 25Mi памяти (запросы), 100m CPU / 100Mi памяти (лимиты).

i

Если в кластере включён Vertical Pod Autoscaler (VPA), управление значениями лимитов происходит автоматически, с помощью VPA. Вы можете задать минимальные и максимальные границы через секцию `resources`.

Пример конфигурации


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: v1
  settings:
    resources:
      dex:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      kubeconfigGenerator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      basicAuthProxy:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      dexAuthenticator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      redis:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"

```

Дополнительные ресурсы

- Документация модуля `vertical-pod-autoscaler`

Авторизация

В Deckhouse Kubernetes Platform авторизация реализована на основе стандартного механизма Role-Based Access Control (RBAC) Kubernetes. Это позволяет гибко управлять правами доступа для различных пользователей, групп и сервисных аккаунтов, обеспечивая безопасность и контроль над операциями в кластере.

Платформа поддерживает две ролевые модели:

- Текущая. Подсистема сквозной авторизации расширяет стандартный RBAC-механизм за счёт пользовательских ресурсов — `ClusterAuthorizationRule` и `AuthorizationRule`.
- Экспериментальная. Эта модель также предполагает использование стандартного механизма RBAC. Доступ настраивается с помощью создания ресурсов `RoleBinding` или `ClusterRoleBinding`.

Обе модели поддерживаются модулем `user-authz`. Выбор модели зависит от требований безопасности и сценариев использования.

Кому и когда выдаются права

Есть два сценария выдачи прав в Deckhouse Kubernetes Platform:

- Выдача прав пользователям для работы через консольные клиенты, веб-интерфейсы и другие инструменты для администрирования, разработки и управления кластером.
- Выдача прав сервисным аккаунтам для автоматизации задач, таких как развёртывание приложений и их обновление (чаще всего при помощи подхода IaC). Примерами таких сервисов могут быть CI/CD-системы, системы мониторинга и другие.

При успешном прохождении аутентификации пользователи и сервисные аккаунты получают права доступа к ресурсам кластера на основе настроек авторизации.

Аутентификация пользователей

В Deckhouse Kubernetes Platform поддерживается несколько способов аутентификации пользователей. Подробнее о них можно узнать в разделе Аутентификация пользователей.

Аутентификация сервисных аккаунтов

Сервисные аккаунты (ServiceAccount) в Kubernetes — это специальные учётные записи, которые используются для автоматизации задач и взаимодействия с API кластеров. Они позволяют приложениям и сервисам безопасно взаимодействовать с Kubernetes API. В Deckhouse Kubernetes Platform сервисные аккаунты для внешних по отношению к кластеру сервисов создаются для единообразия в пространстве имён `d8-service-accounts`.

Пример манифеста для создания ServiceAccount:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: d8-service-accounts
```

После создания ServiceAccount необходимо выписать токен для него, чтобы сервис мог аутентифицироваться в кластере. Для этого создаётся секрет, который содержит токен доступа.

Пример манифеста для создания секрета с токеном ServiceAccount:

```
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: d8-service-accounts
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
```

Дополнительные ресурсы

- Документация модуля `user-authz`

Текущая модель авторизации

Для реализации текущей ролевой модели в кластере должен быть включён модуль `user-authz`. Модуль создаёт набор кластерных ролей (ClusterRole), подходящий для большинства задач по управлению доступом пользователей и групп.



С версии Deckhouse Kubernetes Platform v1.64 в модуле реализована экспериментальная модель ролевого доступа. Текущая модель ролевого доступа продолжит работать, но в будущем будет объявлена устаревшей (deprecated).

Функциональности экспериментальной и текущей моделей ролевого доступа несовместимы. Автоматическая конвертация ресурсов невозможна.

Особенности текущей ролевой модели:

- Реализует role-based-подсистему сквозной авторизации, расширяя функционал стандартного механизма RBAC.
- Настройка прав доступа происходит с помощью кастомных ресурсов ClusterAuthorizationRule и AuthorizationRule.
- Управление доступом к инструментам масштабирования (параметр `allowScale` ресурса ClusterAuthorizationRule или AuthorizationRule).
- Управление доступом к форвардингу портов (параметр `portForwarding` ресурса ClusterAuthorizationRule или AuthorizationRule).
- Управление списком разрешённых пространств имён в формате labelSelector (параметр `namespaceSelector` ресурса ClusterAuthorizationRule).

Высокоуровневые роли, используемые для реализации модели

Для реализации текущей ролевой модели с помощью модуля `user-authz`, кроме использования RBAC, можно использовать удобный набор высокоуровневых ролей:

Роль	Примеры доступных действий	Ограничения
User	Просмотр подов, логов, Deployment	Нет доступа к секретам, портам, контейнерам
PrivilegedUser	Вход в контейнеры (<code>kubectl exec</code>), чтение секретов, удаление подов	Не может изменять Deployment/Service
Editor	Создание/удаление Deployment, Service, ConfigMap	Нет доступа к ReplicaSet, ClusterRoles

Роль	Примеры доступных действий	Ограничения
Admin	Удаление ReplicaSet, управление RBAC в пространстве имён	Нет доступа к ресурсам на уровне кластера
ClusterEditor	Создание DaemonSet, ClusterRole, ClusterXXXMetric, KeepalivedInstance (только тех, что могут понадобиться для прикладных задач)	Не может удалять MachineSets
ClusterAdmin	Полный доступ к ClusterRoleBindings, Machines, OpenstackInstanceClasses	Может повысить свои права
SuperAdmin	Любые действия (включая * в RBAC), но с учетом limitNamespaces	Ограничения только через политики кластера



Режим multitenancy (авторизация по пространству имён) в данный момент реализован по временной схеме и **не гарантирует безопасность**.

В случае, если в ресурсе ClusterAuthorizationRule используется namespaceSelector, параметры limitNamespaces и allowAccessToSystemNamespaces не учитываются.

Если вебхук, который реализовывает систему авторизации, по какой-то причине будет недоступен, опции allowAccessToSystemNamespaces, namespaceSelector и limitNamespaces в кастомных ресурсах перестанут применяться и пользователи будут иметь доступ во все пространства имён. После восстановления доступности вебхука опции продолжают работать.

Список доступа для каждой высокоуровневой роли по умолчанию

Сокращения для verbs:

- read — get, list, watch;
- read-write — get, list, watch, create, delete, deletecollection, patch, update;
- write — create, delete, deletecollection, patch, update.

Роль User:

```
read:
- apiextensions.k8s.io/customresourcedefinitions
- apps/daemonsets
- apps/deployments
- apps/replicasets
- apps/statefulsets
- autoscaling.k8s.io/verticalpodautoscalers
- autoscaling/horizontalpodautoscalers
- batch/cronjobs
- batch/jobs
- configmaps
- discovery.k8s.io/endpointslices
- endpoints
- events
- events.k8s.io/events
- extensions/daemonsets
- extensions/deployments
- extensions/ingresses
- extensions/replicasets
- extensions/replicationcontrollers
- limitranges
- metrics.k8s.io/nodes
- metrics.k8s.io/pods
- namespaces
- networking.k8s.io/ingresses
- networking.k8s.io/networkpolicies
- nodes
- persistentvolumeclaims
- persistentvolumes
- pods
- pods/log
- policy/poddisruptionbudgets
- rbac.authorization.k8s.io/rolebindings
- rbac.authorization.k8s.io/roles
- replicationcontrollers
- resourcequotas
- serviceaccounts
- services
- storage.k8s.io/storageclasses
```

Роль PrivilegedUser (включает все правила из роли User):

```
create:
  - pods/eviction
create,get:
  - pods/attach
  - pods/exec
delete,deletecollection:
  - pods
read:
  - secrets
```

Роль **Editor** (включает все правила из роли **User**, **PrivilegedUser**):

```
read-write:
  - apps/deployments
  - apps/statefulsets
  - autoscaling.k8s.io/verticalpodautoscalers
  - autoscaling/horizontalpodautoscalers
  - batch/cronjobs
  - batch/jobs
  - configmaps
  - discovery.k8s.io/endpointslices
  - endpoints
  - extensions/deployments
  - extensions/ingresses
  - networking.k8s.io/ingresses
  - persistentvolumeclaims
  - policy/poddisruptionbudgets
  - serviceaccounts
  - services
write:
  - secrets
```

Роль **Admin** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**):

```
create,patch,update:
  - pods
delete,deletecollection:
  - apps/replicasets
  - extensions/replicasets
read-write:
  - deckhouse.io/authorizationrules
  - rbac.authorization.k8s.io/rolebindings
  - rbac.authorization.k8s.io/roles
```

Роль **ClusterEditor** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**):

```
read:
  - rbac.authorization.k8s.io/clusterrolebindings
  - rbac.authorization.k8s.io/clusterroles
write:
  - apiextensions.k8s.io/customresourcedefinitions
  - apps/daemonsets
  - extensions/daemonsets
  - storage.k8s.io/storageclasses
```

Роль **ClusterAdmin** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**, **Admin**, **ClusterEditor**):

```
read-write:
  - deckhouse.io/clusterauthorizationrules
write:
  - limitranges
  - namespaces
  - networking.k8s.io/networkpolicies
  - rbac.authorization.k8s.io/clusterrolebindings
  - rbac.authorization.k8s.io/clusterroles
  - rbac.authorization.k8s.io/rolebindings
  - rbac.authorization.k8s.io/roles
  - resourcequotas
```

Вы можете получить дополнительный список правил доступа для роли модуля из кластера (существующие пользовательские правила и нестандартные правила из других модулей Deckhouse) с помощью команды:

```
D8_ROLE_NAME=Editor
d8 k get clusterrole -A -o jsonpath="{range .items[?(@.metadata.annotations.user-
auth\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

Пример AuthorizationRule

Используйте **AuthorizationRule** для установки правил доступа для пользователей внутри определённого пространства имен.

```

apiVersion: deckhouse.io/v1alpha1
kind: AuthorizationRule
metadata:
  name: test-rule
spec:
  accessLevel: Admin
  subjects:
    - kind: Admin
      name: admin@example.com

```

Пример ClusterAuthorizationRule

ClusterAuthorizationRule можно использовать для установки правил доступа для пользователей как на уровне всего кластера, так и на уровне определенных пространств имен.

```

apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
    - kind: User
      name: some@example.com
    - kind: ServiceAccount
      name: gitlab-runner-deploy
      namespace: d8-service-accounts
    - kind: Group
      name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # Опция доступна только при включенном режиме enableMultiTenancy (версия Enterprise Edition).
  allowAccessToSystemNamespaces: false
  # Опция доступна только при включенном режиме enableMultiTenancy (версия Enterprise Edition).
  namespaceSelector:
    labelSelector:
      matchExpressions:
        - key: stage
          operator: In
          values:
            - test
            - review
      matchLabels:
        team: frontend

```

Расширение прав доступа для высокоуровневых ролей

Если требуется добавить права для определённой высокоуровневой роли, создайте ClusterRole с аннотацией `user-authz.deckhouse.io/access-level: <AccessLevel>`.

Пример:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete

```

Дополнительные ресурсы

- Документация модуля `user-authz`

Экспериментальная модель авторизации

Экспериментальная ролевая модель построена на принципе агрегации: она объединяет низкоуровневые роли в более крупные, охватывающие типовые задачи. Это упрощает расширение модели за счёт добавления собственных ролей.

Для реализации экспериментальной ролевой модели в кластере должен быть включен модуль `user-authz`. Модуль создает набор кластерных ролей (ClusterRole), подходящий для большинства задач по управлению доступом пользователей и групп.



С версии Deckhouse Kubernetes Platform (DKP) v1.64 в модуле реализована экспериментальная модель ролевого доступа. Текущая модель ролевого доступа продолжит работать, но в будущем перестанет поддерживаться.

Функциональность экспериментальной и текущей моделей несовместимы. Автоматическая конвертация ресурсов невозможна.

В отличие от текущей ролевой модели DKP, экспериментальная ролевая модель не использует ресурсы ClusterAuthorizationRule и AuthorizationRule. Права доступа настраиваются стандартным способом Kubernetes RBAC: через ресурсы RoleBinding или ClusterRoleBinding, в которых указывается одна из ролей, созданных модулем `user-authz`.

Модуль создаёт специальные агрегированные кластерные роли (ClusterRole). Используя эти роли в RoleBinding или ClusterRoleBinding можно решать следующие задачи:

- Управлять доступом к модулям, относящимся к определённой подсистеме платформы.

Например, чтобы дать возможность пользователю, выполняющему функции сетевого администратора, настраивать *сетевые* модули (например, `cni-cilium`, `ingress-nginx`, `istio` и т. д.), можно использовать в ClusterRoleBinding роль `d8:manage:networking:manager`.

- Управлять доступом к *пользовательским* ресурсам модулей в рамках пространства имён.

Например, использование роли `d8:use:role:manager` в RoleBinding, позволит удалять/создавать/редактировать ресурс PodLoggingConfig в пространстве имён, но не даст доступ к таким

ресурсам на уровне кластера, как ClusterLoggingConfig и ClusterLogDestination модуля `log-shipper`, а также не даст возможность настраивать сам модуль `log-shipper`.

Роли, создаваемые модулем, делятся на два класса:

- Use-роли — для назначения прав пользователям (например, разработчикам приложений) **в конкретном пространстве имён**.
- Manage-роли — для назначения прав администраторам.

⚠ Обратите внимание на особенности настройки комбинированного доступа и совместного использования RoleBinding и ClusterAuthorizationRule (CAR) для одного и того же пользователя при включенном в кластере режиме мультитенантности (параметр `enableMultiTenancy: true`). Подробнее — в документации модуля `user-authz`.

Use-роли

⚠ Use-роль можно использовать только в ресурсе RoleBinding.

Use-роли предназначены для назначения прав пользователю **в конкретном пространстве имён**. Пользователями считаются, например, разработчики, использующие настроенный администратором кластер для развёртывания своих приложений. Таким пользователям не нужно управлять модулями DKP или кластером, но им нужно иметь возможность, например, создавать свои Ingress-ресурсы, настраивать аутентификацию приложений и сбор логов с приложений.

Use-роль определяет права на доступ к namespaced-ресурсам модулей и стандартным namespaced-ресурсам Kubernetes (Pod, Deployment, Secret, ConfigMap и т. п.).

Модуль `user-authz` создаёт следующие use-роли:

Роль	Доступные действия	Ограничения доступа
<code>d8:use:role:viewer</code>	Просмотр Pod, Deployment, Service (кроме секретов и RBAC)	Нет доступа к <code>exec</code> , портам, изменению ресурсов
<code>d8:use:role:user</code>	Чтение секретов, <code>kubectl exec</code> , <code>port-forward</code> , удаление подов, масштабирование реплик	Не может создавать/редактировать объекты
<code>d8:use:role:manager</code>	Создание/изменение Pod, ConfigMap, Deployment, управление ресурсами модулей (например, Certificate)	Нет доступа к Quota и RBAC
<code>d8:use:role:admin</code>	Управление ServiceAccount, Role, ResourceQuota, NetworkPolicy	Полный доступ в пределах пространства имён

Ключевые отличия ролей:

- `viewer` — только `read-only` (без секретов и RBAC);
- `user` — добавляет доступ к секретам, подам и сетевым функциям (`port-forward`);
- `manager` — разрешает управлять ресурсами приложений и связанными ресурсами модулей;
- `admin` — полный контроль над пространством имён (включая RBAC и квоты).

Manage-роли

⚠ Manage-роли не предоставляют доступ к пространствам имён пользовательских приложений.

Manage-роль определяет доступ только к системным пространствам имён (начинающимся с `d8-` или `kube-`), и только к тем из них, в которых работают модули соответствующей подсистемы роли.

Manage-роли предназначены для назначения прав на управление всей платформой или её частью (подсистемой), но не самими приложениями пользователей. С помощью manage-роли можно, например, дать возможность администратору безопасности управлять модулями, ответственными за функции безопасности кластера. Такой администратор сможет управлять аутентификацией, авторизацией и политиками безопасности, но не получит доступ к другим частям кластера (например, сетевой подсистеме или мониторингу) и не сможет менять настройки в пространствах имён приложений.

Manage-роль определяет права на доступ:

- к ресурсам Kubernetes на уровне кластера;
- к управлению модулями DKP (ресурсы ModuleConfig) в рамках подсистемы роли, или всеми модулями DKP для роли `d8:manage:all:*`;
- к управлению ресурсами модулей DKP на уровне кластера в рамках подсистемы роли или всеми ресурсами модулей DKP для роли `d8:manage:all:*`;
- к системным пространствам имён (начинающимся с `d8-` или `kube-`), в которых работают модули подсистемы роли, или ко всем системным пространствам имён для роли `d8:manage:all:*`.

Формат названия manage-роли — `d8:manage:<SUBSYSTEM>:<ACCESS_LEVEL>`, где:

- `SUBSYSTEM` — подсистема роли. Может быть либо одной из подсистем списка, либо `all` для доступа в рамках всех подсистем;
- `ACCESS_LEVEL` — уровень доступа.

Примеры manage-ролей:

- `d8:manage:all:viewer` — доступ на просмотр конфигурации всех модулей DKP (ресурсы `ModuleConfig`), их ресурсов на уровне кластера, их `namespaced`-ресурсов и стандартных объектов Kubernetes (кроме секретов и ресурсов RBAC) во всех системных пространствах имён (начинающихся с `d8-` или `kube-`);
- `d8:manage:all:manager` — аналогично роли `d8:manage:all:viewer`, только доступ на уровне `admin`, т. е. просмотр/создание/изменение/удаление конфигурации всех модулей DKP (ресурсы `ModuleConfig`), их ресурсов на уровне кластера, их `namespaced`-ресурсов и стандартных объектов Kubernetes во всех системных пространствах имён (начинающихся с `d8-` или `kube-`);
- `d8:manage:observability:viewer` — доступ на просмотр конфигурации модулей DKP (ресурсы `ModuleConfig`) из подсистемы `observability`, их ресурсов на уровне кластера, их `namespaced`-ресурсов и стандартных объектов Kubernetes (кроме секретов и ресурсов RBAC) в системных пространствах имён `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, `kube-prometheus-pushgateway`.

Модуль предоставляет два уровня доступа для администратора:

- `viewer` — позволяет просматривать стандартные ресурсы Kubernetes, конфигурацию модулей (ресурсы `ModuleConfig`), ресурсы модулей на уровне кластера и `namespaced`-ресурсы модулей в пространстве имен модуля;
- `manager` — дополнительно к роли `viewer` позволяет управлять стандартными ресурсами Kubernetes, конфигурацией модулей (ресурсы `ModuleConfig`), ресурсами модулей на уровне кластера и `namespaced`-ресурсами модулей в пространстве имен модуля.

Подсистемы ролевой модели

Каждый модуль DKP принадлежит определённой подсистеме. Для каждой подсистемы существует набор ролей с разными уровнями доступа. Роли обновляются автоматически при включении или отключении модуля.

Например, для подсистемы `networking` существуют следующие manage-роли, которые можно использовать в `ClusterRoleBinding`:

- `d8:manage:networking:viewer`;
- `d8:manage:networking:manager`.

Область действия роли зависит от того, к какой подсистеме она принадлежит:

- Область действия ролей из подсистемы `all` — все системные (начинающиеся с `d8-` или `kube-`) неймспейсы кластера.
- Область действия ролей из других подсистем — неймспейсы, в которых работают модули подсистемы (подробнее — в таблице состава подсистем), а также все `cluster-wide` объекты модулей подсистемы.

Состав подсистем ролевой модели

Подсистема	Модули подсистемы	Неймспейсы, используемые модулями подсистемы
<code>all</code>	Все модули	Все неймспейсы
<code>deckhouse</code>	• <code>deckhouse</code> • <code>multitenancy-manager</code>	• <code>d8-multitenancy-manager</code> • <code>d8-system</code>
<code>infrastructure</code>	• <code>chrony</code> • <code>cloud-provider-aws</code> • <code>cloud-provider-azure</code> • <code>cloud-provider-dynamix</code> • <code>cloud-provider-gcp</code> • <code>cloud-provider-huaweicloud</code> • <code>cloud-provider-openstack</code> • <code>cloud-provider-vcd</code> • <code>cloud-provider-vmware</code> • <code>cloud-provider-yandex</code> • <code>cloud-provider-zvirt</code> • <code>descheduler</code> • <code>istio</code> • <code>keepalived</code> • <code>local-path-provisioner</code> • <code>loki</code> • <code>node-manager</code> • <code>terraform-manager</code> • <code>vertical-pod-autoscaler</code>	• <code>d8-chrony</code> • <code>d8-cloud-instance-manager</code> • <code>d8-cloud-provider-aws</code> • <code>d8-cloud-provider-azure</code> • <code>d8-cloud-provider-dynamix</code> • <code>d8-cloud-provider-gcp</code> • <code>d8-cloud-provider-huaweicloud</code> • <code>d8-cloud-provider-openstack</code> • <code>d8-cloud-provider-vcd</code> • <code>d8-cloud-provider-vmware</code> • <code>d8-cloud-provider-yandex</code> • <code>d8-cloud-provider-zvirt</code> • <code>d8-descheduler</code> • <code>d8-istio</code> • <code>d8-keepalived</code> • <code>d8-local-path-provisioner</code> • <code>d8-monitoring</code> • <code>d8-system</code> • <code>kube-system</code>
<code>kubernetes</code>	• <code>control-plane-manager</code> • <code>flow-schema</code> • <code>kube-dns</code> • <code>kube-proxy</code> • <code>priority-class</code>	• <code>d8-system</code> • <code>kube-system</code>

Подсистема	Модули подсистемы	Неймспейсы, используемые модулями подсистемы
networking	- cilium-hubble - cni-cilium - cni-flannel - cni-simple-bridge - flow-schema - ingress-nginx - istio - kube-dns - kube-proxy - metallb - network-gateway - network-policy-engine - node-local-dns - openvpn - static-routing-manager	- d8-cni-cilium - d8-cni-flannel - d8-cni-simple-bridge - d8-ingress-nginx - d8-istio - d8-metallb - d8-network-gateway - d8-openvpn - d8-static-routing-manager - d8-system - kube-system
observability	- dashboard - extended-monitoring - log-shipper - monitoring-applications - monitoring-custom - monitoring-deckhouse - monitoring-kubernetes - monitoring-kubernetes-control-plane - monitoring-ping - okmeter - operator-prometheus - prometheus - prometheus-metrics-adapter - prometheus-pushgateway - upmeter	- d8-log-shipper - d8-monitoring - d8-okmeter - d8-operator-prometheus - d8-upmeter - kube-prometheus-pushgateway
security	- admission-policy-engine - cert-manager - operator-trivy - runtime-audit-engine - secret-copier - user-authn - user-authz	- d8-admission-policy-engine - d8-cert-manager - d8-operator-trivy - d8-runtime-audit-engine - d8-user-authn

Создание новой роли подсистемы

Если текущие подсистемы не соответствуют требованиям ролевого распределения в компании, можно создать новую подсистему, которая будет включать в себя роли из подсистем `deckhouse`, `kubernetes` и модуля `user-authn`.

Для этого создайте следующую роль:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
rules: []
```

Описание лейблов

Созданная роль содержит следующие лейблы:

- `rbac.deckhouse.io/use-role: admin` — указывает, какую роль хук должен использовать при создании use-ролей;
- `rbac.deckhouse.io/kind: manage` — определяет, что роль относится к типу `manage`. **Этот лейбл обязателен**;
- `rbac.deckhouse.io/level: subsystem` — указывает, что роль относится к уровню подсистемы и будет обрабатываться соответствующим образом;
- `rbac.deckhouse.io/subsystem: custom` — задаёт имя подсистемы, за которую отвечает эта роль;
- `rbac.deckhouse.io/aggregate-to-all-as: manager` — позволяет `manage:all`-роли агрегировать эту роль как `manager`.

Описание селекторов агрегации

Секция `aggregationRule` описывает, какие роли и модули агрегируются в данную роль:

- `rbac.deckhouse.io/kind: manage`, `rbac.deckhouse.io/aggregate-to-deckhouse-as: manager` — агрегирует `manage`-роль из подсистем `deckhouse`, `kubernetes`.
- `rbac.deckhouse.io/kind: manage`, `module: user-authn` — агрегирует все правила из модуля `user-authn`.

Таким образом роль получает права от подсистем `deckhouse`, `kubernetes` и от модуля `user-authn`.

i

- Ограничений на имя роли нет, но рекомендуется придерживаться читаемого и унифицированного стиля.
- Use-роли будут автоматически созданы в пространствах имён соответствующих подсистем и модуля. Тип роли определяется по лейблу.

Расширение пользовательской роли

Допустим, в кластере появился новый кластерный CRD-объект `MySuperResource`. Чтобы выдать права на взаимодействие с ним, необходимо дополнить существующую `manage`-роль, описанную выше.

1. Сначала дополните роль новым селектором:

```
rbac.deckhouse.io/kind: manage
rbac.deckhouse.io/aggregate-to-custom-as: manager
```

Этот селектор позволяет включать роль в агрегирующую роль подсистемы через соответствующий лейбл. После добавления нового селектора роль будет выглядеть так:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-custom-as: manager
  rules: []
```

2. Создайте новую роль, в которой следует определить права для нового ресурса. Например, только чтение:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-custom-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Роль дополнит своими правами роль подсистемы, дав права на просмотр нового объекта.

i Имя роли можно задать произвольно, но для удобства рекомендуется придерживаться принятого стиля.

Расширение существующих manage-subsystem-ролей

Если необходимо расширить существующую роль, выполните те же шаги, что и в пункте выше, но изменив лейблы и название роли.

Пример для расширения роли менеджера из подсистемы `deckhouse (d8:manage:deckhouse:manager)`:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustommodule:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Таким образом новая роль расширит роль `d8:manage:deckhouse`.

Расширение manage-subsystem-ролей с добавлением нового пространства имён

Если необходимо добавить новое пространство имён (для создания в нём use-роли с помощью хука), потребуется добавить лишь один лейбл:

```
"rbac.deckhouse.io/namespace": namespace
```

Этот лейбл сообщает хуку, что в этом пространстве имён нужно создать use-роль:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/namespace: namespace
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

Хук отслеживает ресурсы ClusterRoleBinding и при их создании проверяет manage-роли, чтобы найти все агрегированные роли по правилу aggregationRule. Из каждой из них он извлекает пространство имён из лейбла rbac.deckhouse.io/namespace и создаёт use-роль в этом пространстве.

Расширение существующих use-ролей

Если ресурс принадлежит пространству имён, необходимо расширить use-роль вместо manage-роли. Разница лишь в лейблах и имени:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-kubernetes-as: user
    rbac.deckhouse.io/kind: use
  name: custom:use:capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

Эта роль дополнит роль d8:use:role:user:kubernetes.

Дополнительные ресурсы

- Документация модуля cni-cilium
- Документация модуля ingress-nginx
- Документация модуля istio
- Кастомные ресурсы модуля log-shipper
- Документация модуля user-authn
- Документация модуля user-authz

Модель административного доступа к кластеру

Deckhouse Kubernetes Platform (DKP) поддерживает размещение нескольких файлов kubeconfig на master-узлах (поддержка реализуется модулем control-plane-manager). Понимание их назначения важно для безопасного администрирования кластера.

Файлы kubeconfig на master-узлах

На master-узлах размещаются следующие файлы kubeconfig:

Файл	Идентификация	Назначение
/etc/kubernetes/admin.conf	kubernetes-admin (группа kubeadm:cluster-admins)	Машинный kubeconfig для внутренних операций kubeadm (join, обновление). При включённом модуле user-authz RBAC использует user-authz:cluster-admin и дополнительную ClusterRole. При выключенном user-authz группа привязана к встроенной роли cluster-admin

Файл	Идентификация	Назначение
/etc/kubernetes/super-admin.conf	kubernetes-super-admin (группа system:masters)	Аварийный доступ (break-glass). Обходит RBAC полностью. Ограничьте доступ к файлу сценариями восстановления
/etc/kubernetes/controller-manager.conf	system:kube-controller-manager	Используется kube-controller-manager
/etc/kubernetes/scheduler.conf	system:kube-scheduler	Используется kube-scheduler

Административный доступ на основе RBAC

Начиная с Kubernetes 1.29, kubeadm генерирует admin.conf с группой kubeadm:cluster-admins вместо system:masters. Это обеспечивает управляемый через RBAC административный доступ, который может быть отозван путём удаления ClusterRoleBinding kubeadm:cluster-admins (или нескольких привязок).

Если модуль user-authz **выключен**, DKP привязывает группу kubeadm:cluster-admins к встроенной роли cluster-admin с wildcard-правами (как в обычном кластере kubeadm без дополнительной настройки RBAC).

Если модуль user-authz **включён**, группа привязывается к user-authz:cluster-admin, а вторая ClusterRoleBinding добавляет роль d8:control-plane-manager:admin-kubeconfig-supplement (правила сверх высокоуровневой роли, например, для сертификатов и компонентов control plane). Вместе они заменяют одну wildcard-роль cluster-admin для этой идентичности. Для полного неограниченного доступа используйте super-admin.conf.

Рекомендуемый административный доступ

Если модуль user-authn включён, используйте персонализированный kubeconfig на основе OIDC, получаемый через kubeconfig-генератор. Это обеспечивает индивидуальную ответственность и журнал аудита.

Если user-authn отключён, вы можете явно использовать административный kubeconfig на master-узле:

```
d8 k --kubeconfig=/etc/kubernetes/admin.conf <команда>
```

Символическая ссылка root kubeconfig

По умолчанию модуль control-plane-manager создаёт символическую ссылку /root/.kube/config → /etc/kubernetes/admin.conf на master-узлах, что позволяет root-пользователю запускать d8 k без указания --kubeconfig.

Если модуль user-authz включён, это поведение можно отключить, задав rootKubeconfigSymlink: false в конфигурации модуля control-plane-manager:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

Если модуль user-authz выключен, control-plane-manager не использует параметр rootKubeconfigSymlink и сохраняет поведение по умолчанию (символическая ссылка создаётся).

При отключении символической ссылки (при включённом user-authz) ссылка удаляется, если она указывала на admin.conf. Используйте персонализированные учётные данные или явно указывайте --kubeconfig.

Усиление безопасности

Модуль control-plane-manager автоматически ограничивает права доступа к файлам admin.conf и super-admin.conf до 0600 (чтение/запись только для владельца) при каждом цикле согласования. Это предотвращает несанкционированный доступ к этим конфиденциальным учётным данным.

Аварийный доступ (break-glass) к кластеру

В экстренных ситуациях (например, при ошибках в конфигурации RBAC, отказе вебхуков) используйте super-admin.conf:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <команда>
```

Эти учётные данные обходят все проверки RBAC. Используйте их только в крайнем случае и ограничьте круг лиц с доступом к файлу super-admin.conf.

Дополнительные ресурсы

- Документация модуля control-plane-manager

- Документация модуля `user-authn`
- Документация модуля `user-authz`

Выдача прав пользователям и сервисным аккаунтам

Для выдачи прав в Deckhouse Kubernetes Platform в пользовательских ресурсах указывается блок `subjects`.

Для пользователя он указывается в формате:

```
subjects:
- kind: User
  name: <email пользователя>
```



В случае использования модуля `user-authn` и статических пользователей, указывайте в `subjects` именно email пользователя, а не имя ресурса `User`.

или

```
subjects:
- kind: Group
  name: <группа, в которой состоит пользователь>
```

Для сервисного аккаунта блок `subjects` указывается в формате:

```
subjects:
- kind: ServiceAccount
  name: <имя сервисного аккаунта>
  namespace: <пространство имён, в котором создан сервисный аккаунт>
```

Предоставление прав с помощью `AuthorizationRule` и `ClusterAuthorizationRule` (текущая ролевая модель)

При использовании текущей ролевой модели в Deckhouse Kubernetes Platform для предоставления прав пользователям можно использовать ресурсы `AuthorizationRule` и `ClusterAuthorizationRule`.

Предоставление прав пользователю в рамках одного пространства имен

Если нужно предоставить права пользователю в рамках одного пространства имен, используйте ресурс `AuthorizationRule`. Он действует в рамках одного пространства имен. Пример:

```
apiVersion: deckhouse.io/v1
kind: AuthorizationRule
metadata:
  name: dev-access
  namespace: dev-namespace
spec:
  subjects:
  - kind: User
    name: dev-user@example.com
  accessLevel: Admin
  portForwarding: true
```

Предоставление прав пользователю во всех пространствах имен

Если нужно предоставить права пользователю во всех пространствах имен, включая системные (например, для предоставления прав администратора), используйте ресурс `ClusterAuthorizationRule`. Он действует во всем кластере.

При необходимости можно ограничить область действия прав, предоставляемых с помощью `ClusterAuthorizationRule`, одним или несколькими пространствами имен. Для этого в его манифесте укажите соответствующие ограничения (но, если позволяет возможность, рекомендуемый вариант для этого — использование `AuthorizationRule`). Пример:

```

apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin-access
spec:
  subjects:
    - kind: User
      name: dev-user@example.com
  # Опция доступна только при включенном режиме enableMultiTenancy
  # в модуле user-authz (версия Enterprise Edition).
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
  accessLevel: SuperAdmin
  portForwarding: true

```

Предоставление прав с помощью ClusterRoleBinding и RoleBinding (экспериментальная ролевая модель)

При использовании экспериментальной ролевой модели в Deckhouse Kubernetes Platform для предоставления прав пользователям можно использовать ресурсы ClusterRoleBinding и RoleBinding.

Назначение прав администратору кластера (экспериментальная ролевая модель)

Для назначения прав администратору кластера используйте manage-роль `d8:manage:all:manager` в ClusterRoleBinding.

Пример назначения прав администратору кластера (User `jane`):

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
  - kind: User
    name: jane.doe@example.com
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io

```

Права, которые получит пользователь

Права, которые получит пользователь, будут ограничены рамками пространств имён, начинающихся с `d8-` или `kube-`.

Пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание ресурсов Kubernetes и модулей DKP.
- Изменение конфигурации модулей (просмотр, изменение, удаление и создание ресурсов ModuleConfig).
- Выполнение следующих команд к подам и сервисам:
 - `kubectl attach`;
 - `kubectl exec`;
 - `kubectl port-forward`;
 - `kubectl proxy`.

Назначение прав сетевому администратору (экспериментальная ролевая модель)

Для назначения прав сетевому администратору на управление сетевой подсистемой кластера используйте manage-роль `d8:manage:networking:manager` в ClusterRoleBinding.

Пример назначения прав сетевому администратору (User `jane`):

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:networking:manager
  apiGroup: rbac.authorization.k8s.io

```

Список прав, которые получит пользователь

Права, которые получит пользователь, будут ограничены следующим списком пространств имён модулей DKP из подсистемы `networking` (фактический список зависит от списка включённых в кластере модулей):

- `d8-cni-cilium`;
- `d8-cni-flannel`;
- `d8-cni-simple-bridge`;
- `d8-ingress-nginx`;
- `d8-istio`;
- `d8-metallb`;
- `d8-network-gateway`;
- `d8-openvpn`;
- `d8-static-routing-manager`;
- `d8-system`;
- `kube-system`.

Пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание *стандартных* ресурсов Kubernetes в пространстве имён модулей из подсистемы `networking`.

Пример ресурсов, которыми сможет управлять пользователь (список не полный):

- `Certificate`;
 - `CertificateRequest`;
 - `ConfigMap`;
 - `ControllerRevision`;
 - `CronJob`;
 - `DaemonSet`;
 - `Deployment`;
 - `Event`;
 - `HorizontalPodAutoscaler`;
 - `Ingress`;
 - `Issuer`;
 - `Job`;
 - `Lease`;
 - `LimitRange`;
 - `NetworkPolicy`;
 - `PersistentVolumeClaim`;
 - `Pod`;
 - `PodDisruptionBudget`;
 - `ReplicaSet`;
 - `ReplicationController`;
 - `ResourceQuota`;
 - `Role`;
 - `RoleBinding`;
 - `Secret`;
 - `Service`;
 - `ServiceAccount`;
 - `StatefulSet`;
 - `VerticalPodAutoscaler`;
 - `VolumeSnapshot`.
- Просмотр, изменение, удаление и создание ресурсов в пространстве имён модулей из подсистемы `networking`.

Список ресурсов, которыми сможет управлять пользователь:

- `EgressGateway`;
 - `EgressGatewayPolicy`;
 - `FlowSchema`;
 - `IngressClass`;
 - `IngressIstioController`;
 - `IngressNginxController`;
 - `IPRuleSet`;
 - `IstioFederation`;
 - `IstioMulticluster`;
 - `RoutingTable`.
- Изменение конфигурации модулей (просмотр, изменение, удаление и создание ресурсов `ModuleConfig`) из

подсистемы `networking`.

Список модулей, которыми сможет управлять пользователь:

- `cilium-hubble`;
 - `cni-cilium`;
 - `cni-flannel`;
 - `cni-simple-bridge`;
 - `flow-schema`;
 - `ingress-nginx`;
 - `istio`;
 - `kube-dns`;
 - `kube-proxy`;
 - `metallb`;
 - `network-gateway`;
 - `network-policy-engine`;
 - `node-local-dns`;
 - `openvpn`;
 - `static-routing-manager`.
- Выполнение следующих команд к подам и сервисам в пространстве имён модулей из подсистемы `networking`:
 - `kubectl attach`;
 - `kubectl exec`;
 - `kubectl port-forward`;
 - `kubectl proxy`.

Назначение административных прав пользователю в рамках пространства имён (экспериментальная ролевая модель)

Чтобы назначить/ограничить права пользователя конкретными пространствами имён, используйте в `RoleBinding` `use-роль` с соответствующим уровнем доступа.

Например, для назначения прав на управление ресурсами приложений в рамках пространства имён, но без возможности настройки модулей DKP, используйте роль `d8:use:role:admin` в `RoleBinding` в соответствующем пространстве имён.

Пример назначения прав разработчику приложений (User `app-developer`) в пространстве имён `myapp`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:use:role:admin
  apiGroup: rbac.authorization.k8s.io
```

Список прав, которые получит пользователь

В рамках пространства имён `myapp` пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание ресурсов Kubernetes. Например, следующих ресурсов:
 - `Certificate`;
 - `CertificateRequest`;
 - `ConfigMap`;
 - `ControllerRevision`;
 - `CronJob`;
 - `DaemonSet`;
 - `Deployment`;
 - `Event`;
 - `HorizontalPodAutoscaler`;
 - `Ingress`;
 - `Issuer`;
 - `Job`;
 - `Lease`;
 - `LimitRange`;
 - `NetworkPolicy`;
 - `PersistentVolumeClaim`;
 - `Pod`;
 - `PodDisruptionBudget`;
 - `ReplicaSet`;
 - `ReplicationController`;
 - `ResourceQuota`;
 - `Role`;
 - `RoleBinding`;
 - `Secret`;

- Service;
 - ServiceAccount;
 - StatefulSet;
 - VerticalPodAutoscaler;
 - VolumeSnapshot.
- Просмотр, изменение, удаление и создание следующих ресурсов модулей DKP:
 - DexAuthenticator;
 - DexClient;
 - PodLoggingConfig.
- Выполнение следующих команд к подам и сервисам:
 - `kubectl attach`;
 - `kubectl exec`;
 - `kubectl port-forward`;
 - `kubectl proxy`.

Дополнительные ресурсы

- Документация модуля `user-authn`
- Кастомные ресурсы модуля `user-authz`

Проверка наличия прав у пользователя

Чтобы проверить наличие прав доступа у пользователя, выполните следующую команду, в которой будут указаны:

- `resourceAttributes` (как в RBAC) — к чему проверяется доступ;
- `user` — имя пользователя;
- `groups` — группы пользователя.

i При совместном использовании с модулем `user-authn` группы и имя пользователя можно посмотреть в логах Dex с помощью команды `d8 k -n d8-user-authn logs -l app=dex` (видны только при авторизации).

```
cat <<EOF | 2>&1 d8 k create --raw /apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq
.status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

В результате будет видно, есть ли доступ и на основании какой роли.

Пример ответа при наличии прав доступа у пользователя:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by ClusterRoleBinding \"system:kube-controller-manager\" of ClusterRole \"system:kube-controller-manager\" to User \"system:kube-controller-manager\""
```

Пример ответа при отсутствии прав доступа у пользователя:

```
{
  "allowed": false
}
```

Если в кластере включён режим **multitenancy**, выполните ещё одну проверку, чтобы убедиться, что у пользователя есть доступ в пространство имён:

```
cat <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-config.yaml create
--raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

Пример ответа при наличии прав доступа у пользователя:

```
{
  "allowed": false
}
```

Сообщение `"allowed": false` значит, что вебхук не блокирует запрос. В случае блокировки запроса вебхуком вы получите, например, следующее сообщение:

```
{
  "allowed": false,
  "denied": true,
  "reason": "making cluster scoped requests for namespaced resources are not allowed"
}
```

Дополнительные ресурсы

- Документация модуля `user-authn`

Доступ для CI/CD

Для аутентификации CI/CD-пайплайнов в API Kubernetes доступны следующие методы:

- ServiceAccount — токен Kubernetes ServiceAccount.
- Basic Auth — логин и пароль через IdP.
- Token Exchange — обмен токена IdP на токен Dex.

ServiceAccount

Токен ServiceAccount используется напрямую для аутентификации в API. Внешний IdP не требуется. Для ServiceAccount можно создавать долгоживущие токены (без TTL, с возможностью отзыва или ротации) и короткоживущие токены.

При совместном использовании одного ServiceAccount несколькими пайплайнами в audit-логах не будет информации о конкретном пайплайне.

Предварительные требования

Для настройки аутентификации через токен для ServiceAccount должны быть соблюдены следующие требования:

- Доступ к кластеру с правами на создание ServiceAccount и секретов.
- Для внешнего доступа: опубликованный через Ingress API-сервер Kubernetes (для публикации используется параметр `publishAPI`) или прямой доступ к API через VPN.

Создание ServiceAccount и долгоживущего токена

Для создания ServiceAccount и токена используйте команду:

```
d8 k create ns ci-deploy || true

cat <<EOF | d8 k apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: ci-deploy
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: ci-deploy
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

i Секрет типа `kubernetes.io/service-account-token` — устаревший подход. Рекомендуется использовать `TokenRequest` API (`d8 k create token ...`).

Выдача прав

Подробнее о выдаче прав — в разделе «Выдача прав пользователям и сервисным аккаунтам».

Назначьте необходимые для `ServiceAccount` права.

Для текущей ролевой модели используйте `ClusterAuthorizationRule`:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-access
spec:
  subjects:
    - kind: ServiceAccount
      name: gitlab-runner-deploy
      namespace: ci-deploy
  accessLevel: Admin
  portForwarding: true
EOF
```

Доступные уровни: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`.

Для экспериментальной ролевой модели используйте `ClusterRoleBinding`:

```
cat <<EOF | d8 k apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: gitlab-runner-access
subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci-deploy
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
apiGroup: rbac.authorization.k8s.io
EOF
```

Получение URL API и сертификата CA

При использовании `publishAPI`:

```
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
echo "API endpoint: https://${API_HOST}"
```

i Если сертификат API подписан публичным CA (Let's Encrypt), параметр `--certificate-authority` не требуется.

Для приватного CA:

```
d8 k -n d8-user-authn get secret kubernetes-api-ca-key-pair -o jsonpath='{.data.ca\.crt}' | base64 -d > /tmp/ca.crt
```

Создание kubeconfig

Пример генерации kubeconfig для созданного ранее ServiceAccount:

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config

# Для публичного CA уберите --certificate-authority и --embed-certs.
d8 k config set-cluster $CLUSTER_NAME \
  --server=https://${API_HOST} \
  --certificate-authority=/tmp/ca.crt \
  --embed-certs=true \
  --kubeconfig=$FILE_NAME

d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n ci-deploy get secret gitlab-runner-deploy-token -o jsonpath='{.data.token}' | base64 -d) \
  --kubeconfig=$FILE_NAME

d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME

d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

Короткоживущие токены (TokenRequest)

TokenRequest API позволяет создавать токены с ограниченным сроком действия. При создании такого токена укажите в команде имя ServiceAccount, которому нужно предоставить доступ (в примере — `gitlab-runner-deploy`):

```
d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h
```

⚠ Команда `d8 k create token` сама требует доступа к Kubernetes API. Если CI-задача (job) не имеет начального доступа к кластеру, выполнить TokenRequest внутри задачи невозможно.

Типовые сценарии к использованию в CI

Bootstrap-токен с минимальными правами. CI хранит долгоживущий kubeconfig/токен, который имеет право только на создание TokenRequest для конкретного ServiceAccount. В начале задачи bootstrap-токен выпускает короткоживущий токен, который используется для деплоя.

Внешний token broker. Оператор, секрет-менеджер (Vault, External Secrets) или внешний сервис периодически выпускает короткоживущие токены и помещает их в CI-переменные или секреты. CI-задача только использует готовый токен.

i Bootstrap-токен должен иметь минимальные права: только `create` на ресурс `serviceaccounts/token` для конкретного ServiceAccount/namespace. Не используйте `cluster-admin`. Ограничивайте TTL выпускаемых токенов и оперативно отзывайте bootstrap-доступ при его компрометации.

Пример использования без файла kubeconfig:

```
export KUBE_SERVER="https://${API_HOST}"
export KUBE_TOKEN=$(d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h)
d8 k --server=$KUBE_SERVER --token=$KUBE_TOKEN get ns
```

Basic Auth

Аутентификация по логину и паролю через IdP (LDAP, OIDC).



Пароль передаётся в DKP и проверяется через basic-auth-proxy/Dex.



Только один DexProvider в кластере может иметь `enableBasicAuth: true`.

Предварительные требования

Для настройки Basic Auth должны быть соблюдены следующие требования:

- publishAPI включён.
- DexProvider настроен для IdP.

Включение

В DexProvider добавьте `enableBasicAuth: true`. Пример настройки DexProvider приведён в документации модуля user-authn.

Получение эндпоинта API

Для получения эндпоинта API используйте команду:

```
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
echo "https://${API_HOST}"
```

Проверка

Для проверки доступа используйте команду:

```
curl -q -u "$K8S_USER:$K8S_PASSWORD" "https://${API_HOST}/version"
```

Возможные ошибки:

- 401 — неверные учётные данные или Basic Auth не включён.
- 403 — аутентификация прошла, но RBAC запрещает доступ.

Настройка kubeconfig

Пример настройки kubeconfig для Basic Auth:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://<API_HOST>
users:
- name: basic-auth-user
  user:
    username: "<USERNAME>"
    password: "<PASSWORD>"
contexts:
- name: default
  context:
    cluster: my-cluster
    user: basic-auth-user
current-context: default
```

Использование в GitLab CI

Пример использования Basic Auth в GitLab CI:

```
deploy:
  script:
    - d8 k --server="$K8S_SERVER" --username="$K8S_USER" --password="$K8S_PASSWORD" get ns
```

Переменные `K8S_SERVER`, `K8S_USER`, `K8S_PASSWORD` задаются в настройках CI/CD проекта.

Обмен токена (token exchange)

Пайплайн получает токен от IdP, обменивает его в Dex на токен с `aud=kubernetes` и обращается с ним к API.

i Рекомендуется для GitLab CI и GitHub Actions.

DKP/Dex не получает пароль пользователя. Способ получения `IDP_TOKEN` зависит от IdP: OIDC job token (GitLab/GitHub) или эндпоинт токена IdP (client_credentials).

Предварительные требования

Для настройки Token Exchange должны быть соблюдены следующие требования:

- publishAPI включён.
- DexProvider настроен как **тип OIDC**.

⚠ Token exchange не работает с коннекторами `type: GitLab` или `type: GitHub`. Настройте GitLab/GitHub как OIDC-провайдеры.

Создание DexClient

Для создания DexClient используйте команду:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: ci-token-exchange
  namespace: d8-user-authn
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec: {}
EOF
```

Аннотация `dexclient.deckhouse.io/allow-access-to-kubernetes` позволяет клиенту запрашивать токены с `aud=kubernetes`.

⚠ Предоставляемый таким образом доступ к Kubernetes API действует на уровне всего кластера. По этой причине для добавления аннотации или изменения её значения на `"true"` необходимы права на изменение конфигурации модуля `user-authn` — например, роль `d8:manage:permission:module:user-authn:edit`. Прав на создание DexClient в отдельном неймспейсе недостаточно: admission-контроллер отклонит запрос с сообщением, содержащим имя аннотации.

Добавление аннотации ограничено независимо от указанного значения, включая `"false"`. Это необходимо для совместимости с предыдущими версиями DKP, в которых доступ предоставляется при наличии аннотации независимо от её значения.

Ограничение не распространяется на объект, у которого аннотация уже установлена. Удаление и повторное создание DexClient считается повторным добавлением аннотации. Поэтому GitOps-контроллеру, который пересоздаёт объект вместо его обновления, требуется право `update` на ресурс `moduleconfigs` с именем `user-authn`.

Получение client secret:

```
d8 k -n d8-user-authn get secret dex-client-ci-token-exchange -o jsonpath='{.data.clientSecret}' | base64 -d
```

Формат **client_id**: `dex-client-ci-token-exchange@d8-user-authn`.

Получение URL Dex и API

Для получения URL Dex и API используйте команды:

```
DEX_HOST=$(d8 k -n d8-user-authn get ingress dex -o jsonpath='{.spec.rules[0].host}')
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
```

Выдача RBAC

DKP настраивает kube-apiserver на проверку токенов Dex. Claims `email` и `groups` из токена используются для RBAC.

Набор claims, которые требуются kube-apiserver для аутентификации, зависит от конфигурации. Если kube-apiserver требует `name`, добавьте scope `profile`.

Подробнее о выдаче прав — в разделе «Выдача прав пользователям и сервисным аккаунтам».

Для текущей ролевой модели используйте ClusterAuthorizationRule:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: ci-deployer
spec:
  subjects:
  - kind: User
    name: deployer@example.com
    accessLevel: Admin
EOF
```

Для экспериментальной ролевой модели используйте ClusterRoleBinding:

```
cat <<EOF | d8 k apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ci-deployer
subjects:
- kind: User
  name: deployer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Запрос обмена токена

Пример запроса на обмен токена:

```
RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
-u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
-d "subject_token=${IDP_TOKEN}" \
-d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
-d "connector_id=${CONNECTOR_ID}" \
-d "scope=openid profile email groups audience:server:client_id:kubernetes" \
-d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")

DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
```

Параметры:

- `subject token` — токен от IdP.
- `subject token_type` — `id token` (GitLab/GitHub) или `access_token` (Keycloak).
- `connector_id` — `metadata.name` ресурса DexProvider.
- `scope` — обязательно `audience:server:client_id:kubernetes` и `profile`.

i Ожидаемая `audience` всегда `kubernetes`. Если API возвращает 401 с ошибкой несоответствия `audience`, убедитесь что `scope` содержит `audience:server:client_id:kubernetes`.

GitLab CI

Пример запроса токена для GitLab CI:


```

deploy:
  id_tokens:
    GITLAB_OIDC_TOKEN:
      aud: https://<DEX_HOST>/
  script:
    - |
      RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
        -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
        -H "Content-Type: application/x-www-form-urlencoded" \
        -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
        -d "subject_token=${GITLAB_OIDC_TOKEN}" \
        -d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
        -d "connector_id=gitlab" \
        -d "scope=openid profile email groups audience:server:client_id:kubernetes" \
        -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")
      DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
      d8 k --server="${K8S_SERVER}" --token="${DEX_TOKEN}" auth whoami

```

Keycloak

Получение токена из Keycloak:

```

KEYCLOAK_TOKEN=$(curl -q -s -X POST "https://<KEYCLOAK_HOST>/realms/<REALM>/protocol/openid-connect/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=client_credentials" \
  -d "client_id=<KEYCLOAK_CLIENT_ID>" \
  -d "client_secret=<KEYCLOAK_CLIENT_SECRET>" | jq -r '.access_token')

```

Обмен в Dex (для Keycloak используется `subject_token_type=access_token`):

i Для обмена `access_token` в DexProvider требуется `getUserInfo: true`.

```

RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
  -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
  -d "subject_token=${KEYCLOAK_TOKEN}" \
  -d "subject_token_type=urn:ietf:params:oauth:token-type:access_token" \
  -d "connector_id=keycloak" \
  -d "scope=openid profile email groups audience:server:client_id:kubernetes" \
  -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")

DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
d8 k --server="https://${API_HOST}" --token="${DEX_TOKEN}" auth whoami

```

Устранение неполадок

i Используйте `d8 k auth whoami` для проверки под какой идентичностью выполняется запрос.

Флаг `--token` может не работать — `kubeconfig` с `auth-provider`, `exec` или `tokenFile` переопределяет токен из флага. Для отладки используйте `-v=8`:

```
d8 k --server="https://${API_HOST}" --token="${DEX_TOKEN}" auth whoami -v=8
```

В выводе будет видно какой Authorization заголовок отправляется. Для надёжной работы с токеном используйте минимальный `kubeconfig` только с `server` и `token`, без `auth-provider/exec`.

Dex 400 — неверный `subject_token`, `subject_token_type` или `connector_id`.

Dex 401 — неверные учётные данные клиента или невалидный `subject token`.

API 401 — токен не прошёл проверку. Проверьте:

- Аннотация `dexclient.deckhouse.io/allow-access-to-kubernetes` на `DexClient`. Если её нет, проверьте, не отклонил ли `admission`-контроллер запрос, который должен был её добавить, и не пересоздал ли `GitOps`-контроллер объект без неё. Добавить аннотацию может только субъект, которому разрешено обновлять `ModuleConfig user-authn`.
- Scope содержит `audience:server:client_id:kubernetes` и `profile`.
- Синхронизация времени между CI-раннером и кластером.

API 403 — аутентификация прошла, но RBAC не разрешает доступ для пользователя или группы из токена.

Декодирование токена для проверки claims:

```
echo "${DEX_TOKEN}" | cut -d. -f2 | base64 -d 2>/dev/null | jq .
```

Дополнительные ресурсы

- Документация модуля `user-authn`
- Кастомные ресурсы модуля `user-authz`

Интеграция с IaaS

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства для интеграции с различными облачными провайдерами и системами виртуализации. Эти средства позволяют:

- автоматически использовать облачную инфраструктуру для заказа виртуальных машин и подключения их к кластеру;
- разворачивать кластеры в облачных средах;
- масштабировать ресурсы при необходимости;
- автоматически создавать базовые инфраструктурные ресурсы (сети, маршруты и т.п.), если это поддерживается выбранным провайдером.

Порядок действий для работы с различными облачными провайдерами одинаков, отличаются только шаги подготовки (создание в облаке сервисного аккаунта и т.д.) и содержимое конфигурационных файлов.

Поддерживаемые облачные провайдеры:

- Yandex Cloud;
- VK Cloud;
- Selectel Cloud;
- OVH Cloud;
- OpenStack;
- Amazon Web Services (AWS);
- Microsoft Azure;
- Google Cloud Platform (GCP).

Возможна интеграция с частными облаками, развёрнутыми с использованием следующих решений:

- VK Cloud;
- OpenStack;
- Huawei Cloud;
- Basis Dynamix.

Помимо облачных провайдеров поддерживается интеграция со следующими системами виртуализации:

- Deckhouse Virtualization Platform (DVP);
- VMware Cloud Director;
- VMware vSphere;
- zVirt.

Также предусмотрена возможность настроить гибридные кластеры.

Публичные облачные провайдеры

Deckhouse Kubernetes Platform (DKP) может быть развёрнута в различных публичных облаках, что позволяет использовать преимущества масштабируемой инфраструктуры и гибкой интеграции с облачными сервисами.

Платформа поддерживает работу с такими провайдерами, как:

- Yandex Cloud;
- VK Cloud;
- Selectel Cloud;
- OVH Cloud;
- OpenStack;
- Amazon Web Services (AWS);
- Microsoft Azure;
- Google Cloud Platform (GCP).

Это упрощает выбор среды для развёртывания и делает возможным использование единых подходов к управлению Kubernetes-кластерами в разных облачных экосистемах.

Подключение и авторизация в Yandex Cloud

Для того чтобы Deckhouse Kubernetes Platform (DKP) мог управлять ресурсами в Yandex Cloud, необходимо:

- создать сервисный аккаунт;
- назначить ему необходимые IAM-роли;
- сгенерировать ключ авторизации;
- при необходимости — зарезервировать публичный IP-адрес;
- обеспечить достаточные квоты на ресурсы в облаке.

Ниже приведены пошаговые действия по настройке подключения.

Подготовка окружения



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

Чтобы проверить, запущены ли службы, выполните команды:

```
systemctl status cloud-config.service
systemctl status cloud-final.service
systemctl status cloud-init.service
```

Создание сервисного аккаунта

Для управления ресурсами в Yandex Cloud через DKP создайте сервисный аккаунт и выдайте ему права на редактирование. Подробную инструкцию по созданию сервисного аккаунта смотрите в документации Yandex Cloud.

Для создания сервисного аккаунта выполните команду:

```
yc iam service-account create --name deckhouse
```

Команда вернёт информацию о созданном сервисном аккаунте:

```
id: <userID>
folder_id: <folderID>
created_at: "YYYY-MM-DDTHH:MM:SSZ"
name: deckhouse
```



Сохраните `userID` и `folderID` — они понадобятся в следующих шагах.

Назначение IAM-ролей

Для работы DKP с ресурсами облака назначьте сервисному аккаунту следующие роли:

```
yc resource-manager folder add-access-binding --id <folderID> --role compute.editor --subject serviceAccount:<userID>
yc resource-manager folder add-access-binding --id <folderID> --role vpc.admin --subject serviceAccount:<userID>
yc resource-manager folder add-access-binding --id <folderID> --role load-balancer.editor --subject serviceAccount:<userID>
```

Генерация авторизованного ключа

Создайте JSON-файл с авторизацией для использования в конфигурации:

```
yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
```

Содержимое файла `deckhouse-sa-key.json` используйте в поле `provider.serviceAccountJSON` при описании конфигурации кластера.

Проверка и увеличение квот

Убедитесь, что облачный аккаунт имеет необходимые квоты для развёртывания и масштабирования:

- Виртуальные процессоры: 64
- Объём SSD-дисков: 2000 ГБ
- Количество ВМ: 25
- Объём RAM: 256 ГБ

Увеличьте квоты через консоль Yandex Cloud, если необходимо.

Резервирование публичного IP

Если используется схема размещения WithoutNAT или WithNATInstance, и требуется фиксированный внешний IP-адрес (например, для указания в `externalIPAddresses`, `natInstanceExternalAddress` или для bastion-хоста), выполните команду:

```
yc vpc address create --external-ipv4 zone=ru-central1-a
```

Пример вывода команды:

```
id: e9b4cfmmnc1mhgij75n7
folder_id: blgog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
requirements: {}
reserved: true
```

После выполнения этих шагов у вас будут все необходимые данные для формирования ресурса YandexClusterConfiguration, описывающего кластер в Yandex Cloud.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-yandex`

Схемы размещения и настройка Yandex Cloud

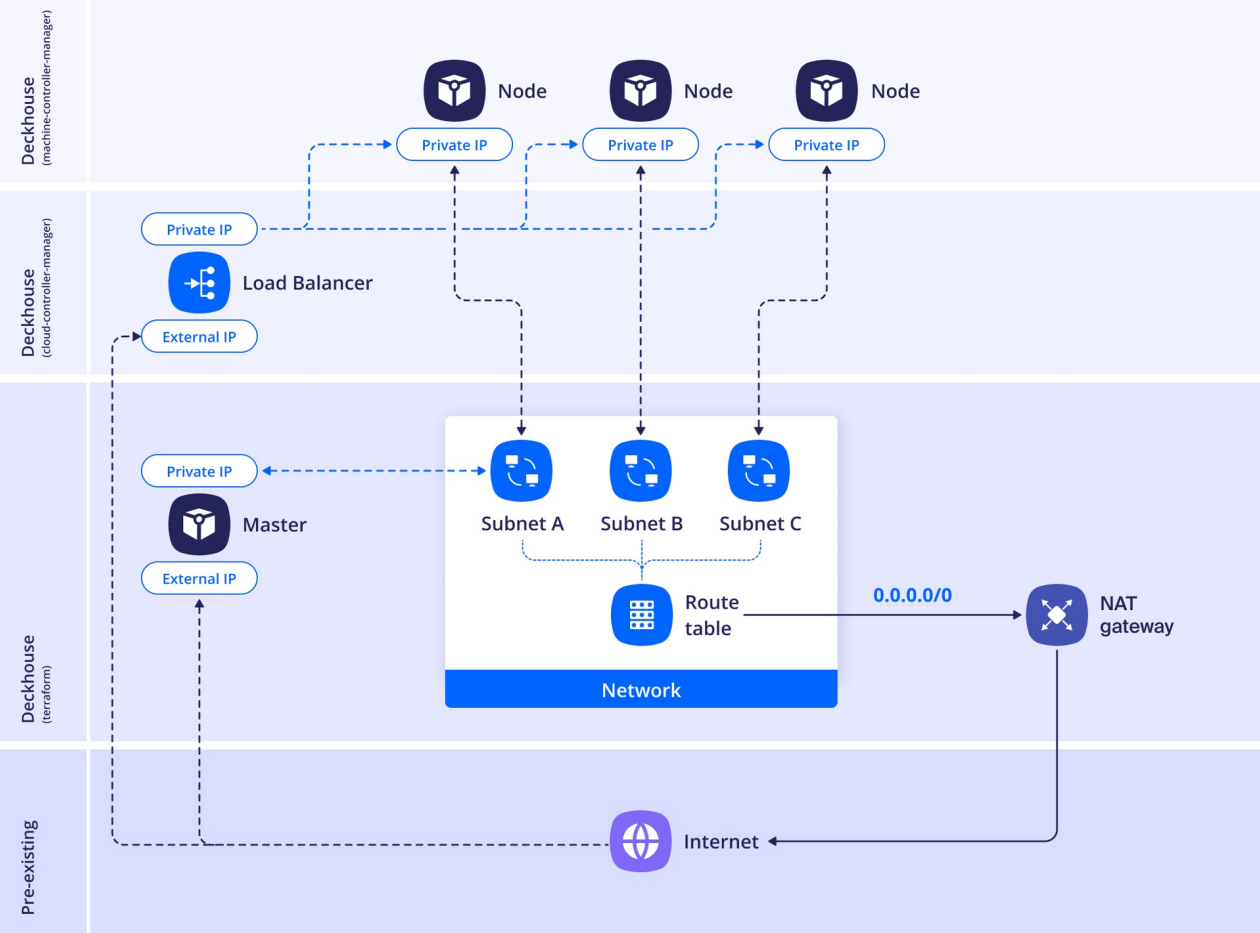
Схемы размещения

Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре Yandex Cloud и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

✖ В данной схеме размещения узлы не будут иметь публичных IP-адресов и будут выходить в интернет через NAT-шлюз (NAT Gateway) Yandex Cloud. NAT-шлюз (NAT Gateway) использует случайные публичные IP-адреса из выделенных диапазонов. Из-за этого невозможно добавить в белый список (whitelist) адреса облачных ресурсов, находящихся за конкретным NAT-шлюзом, на стороне других сервисов.

⚠ Так как в данной схеме размещения узлы создаются без публичных IP-адресов, для установки кластера на master-узел должен быть SSH-доступ с машины, на которой запускается dhctl: напрямую по приватной сети или через bastion-хост. Если master-узел недоступен напрямую, запускайте установку с параметрами `--ssh-bastion-host`, `--ssh-bastion-user` и, при необходимости, `--ssh-bastion-port`.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIiwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...lZPJJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  zones:
    - ru-central1-a
    - ru-central1-b
    - ru-central1-d
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    # Опционально: раскомментируйте, чтобы назначить master-узлам публичные IP-адреса.
    # externalIPAddresses:
    # - "<ZONE_A_EXTERNAL_IP_MASTER_1>"
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    # - <ZONE_D_SUBNET_ID>
    additionalLabels:
      takes: priority
nodeGroups:
- name: worker
  replicas: 2
  zones:
    - ru-central1-a
    - ru-central1-b
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    # Опционально: раскомментируйте, чтобы назначить worker-узлам публичные IP-адреса.
    # externalIPAddresses:
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    additionalLabels:
      role: example
labels:
  billing: prod
dhcpOptions:
  domainName: test.local
  domainNameServers:
    - <DNS_SERVER_1>
    - <DNS_SERVER_2>

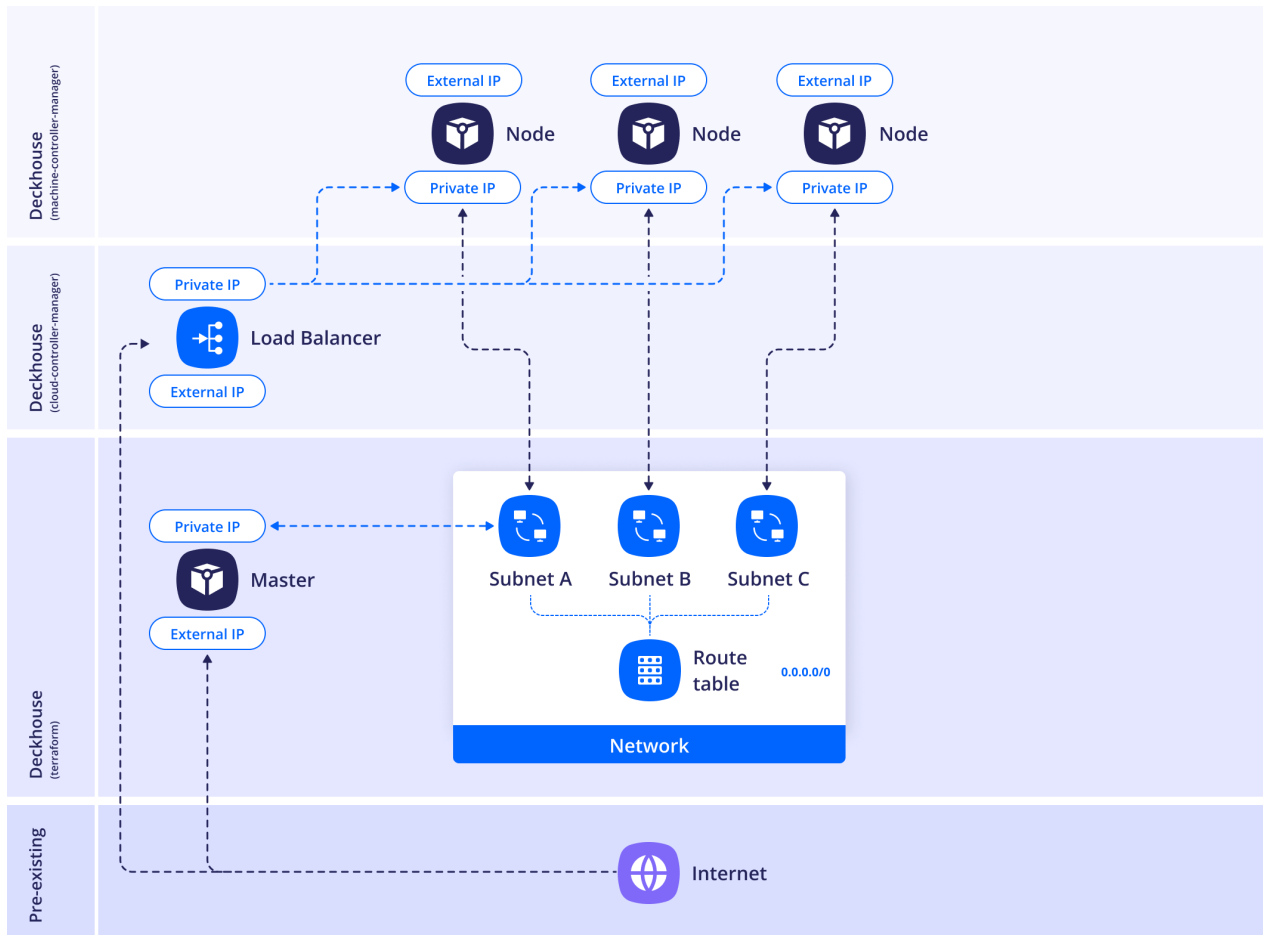
```

WithoutNAT

В данной схеме размещения NAT (любого вида) не используется, а каждому узлу выдается публичный IP-адрес.



В DKP нет поддержки групп безопасности (security group), поэтому все узлы кластера будут доступны без ограничения подключения.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    externalIPAddresses:
      - "Auto"
      - "Auto"
      - "Auto"
    externalSubnetIDs:
      - <ZONE_A_SUBNET_ID>
      - <ZONE_B_SUBNET_ID>
      - <ZONE_D_SUBNET_ID>
    zones:
      - ru-central1-a
      - ru-central1-b
      - ru-central1-d
nodeGroups:
- name: worker
  replicas: 2
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    externalIPAddresses:
      - "<ZONE_A_EXTERNAL_IP_WORKER_1>"
      - "Auto"
    externalSubnetIDs:
      - <ZONE_A_SUBNET_ID>
      - <ZONE_B_SUBNET_ID>
    zones:
      - ru-central1-a
      - ru-central1-b
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
    - <DNS_SERVER_1>
    - <DNS_SERVER_2>

```

WithNATInstance

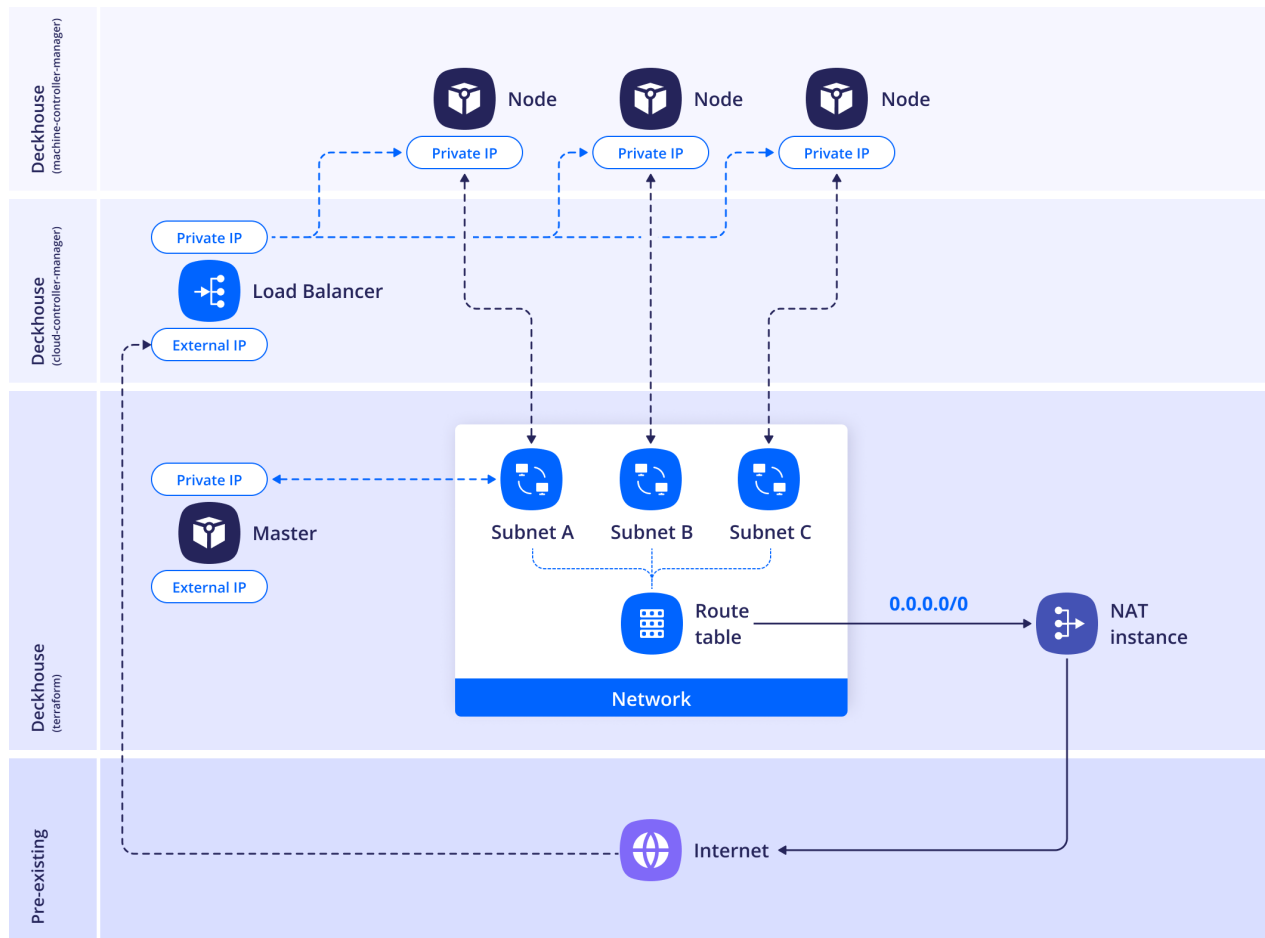
В данной схеме размещения в отдельной подсети создается NAT-инстанс, а в таблицу маршрутизации подсетей зон добавляется правило с маршрутом на 0.0.0.0/0 с NAT-инстансом в качестве nexthop'a. Подсеть выделяется для предотвращения петли маршрутизации и не должна пересекаться с другими сетями, используемыми в кластере.

Для размещения NAT-инстанса в существующей подсети используйте параметр `withNATInstance.internalSubnetID` — инстанс будет создан в зоне, соответствующей этой подсети.

Если необходимо создать новую подсеть, укажите параметр `withNATInstance.internalSubnetCIDR` — в ней будет размещён NAT-инстанс.

i Обязателен один из параметров: `withNATInstance.internalSubnetID` или `withNATInstance.internalSubnetCIDR`.

Если также указан `withNATInstance.externalSubnetID`, NAT-инстанс будет подключён к этой подсети через вторичный сетевой интерфейс.



Пример конфигурации схемы размещения:


```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithNATInstance
withNATInstance:
  natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
  internalSubnetID: <INTERNAL_SUBNET_ID>
  externalSubnetID: <EXTERNAL_SUBNET_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIEwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 1
  zones:
    - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
nodeGroups:
  - name: worker
    replicas: 1
    zones:
      - ru-central1-a
    instanceClass:
      cores: 4
      memory: 8192
      imageID: <IMAGE_ID>
      coreFraction: 50
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
    - <DNS_SERVER_1>
    - <DNS_SERVER_2>

```

i

В примере master и worker-узлы создаются без публичных IP-адресов и без дополнительных внешних сетевых интерфейсов. Исходящий трафик из подсетей кластера проходит через NAT-инстанс.

NAT-инстанс не используется инсталлятором `dhctl` автоматически в качестве bastion-хоста или jump-хоста. Для установки кластера на master-узел должен быть SSH-доступ с машины, на которой запускается `dhctl`: напрямую или через bastion-хост.

Если master-узел не имеет публичного IP-адреса или недоступен напрямую, запускайте установку с параметрами `--ssh-bastion-host` и `--ssh-bastion-user`.

Если необходимо подключаться к master-узлу напрямую по публичному IP-адресу, укажите для него параметры `externalIPAddresses` и `externalSubnetIDs`. Учитывайте, что интерфейс из сети, указанной в `externalSubnetIDs`, будет использоваться как шлюз по умолчанию для узла.

Назначение YandexClusterConfiguration

Для интеграции Deckhouse Kubernetes Platform с Yandex Cloud необходимо описать инфраструктуру кластера с помощью ресурса `YandexClusterConfiguration`.

`YandexClusterConfiguration` — это объект Custom Resource (CR), описывающий параметры интеграции с облаком Yandex Cloud. Он используется DKP для:

- размещения master и worker-узлов в облаке;
- задания схемы сетевого взаимодействия;
- подключения к API Yandex Cloud с использованием JSON-ключа авторизации;
- настройки подсетей, публичных IP, ресурсов виртуальных машин и др.

Обязательные поля:

- `apiVersion` — должен быть `deckhouse.io/v1`;
- `kind` — всегда `YandexClusterConfiguration`.

Пример заголовка ресурса:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
```

Чтобы отредактировать этот ресурс в работающем кластере, выполните команду:

```
d8 system edit provider-cluster-configuration
```

После внесения изменений их необходимо применить с помощью команды:

```
dhctl converge
```

Пример конфигурации

Ниже приведён пример минимальной конфигурации ресурса `YandexClusterConfiguration`, описывающей кластер с одной master-группой и одной группой рабочих узлов. Конфигурация использует схему размещения `Standard`, заданы базовые параметры вычислительных ресурсов, публичный ключ SSH, а также идентификаторы облака и каталога:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
nodeNetworkCIDR: 127.0.0.1/8
labels:
  label-2: b
sshPublicKey: "<SSH_PUBLIC_KEY>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd8nb7ecsbvj76dfaa8b
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd8nb7ecsbvj76dfaa8b
    coreFraction: 50
    externalIPAddresses:
    - 198.51.100.5
    - Auto
provider:
  cloudID: "<CLOUD_ID>"
  folderID: "<FOLDER_ID>"
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJt+ \n-----END PRIVATE KEY-----\n"
    }
```

- Master-группа состоит из одного узла без явного указания зоны.
- Worker-группа содержит один узел, размещённый в зоне `ru-central1-a`, с двумя внешними IP-адресами: один задан вручную (`198.51.100.5`), второй заказан автоматически (`Auto`).
- Указан ключ `serviceAccountJSON`, необходимый для подключения к API Yandex Cloud.
- Используется CIDR-подсеть `127.0.0.1/8` и добавлен label `label-2: b` на уровне кластера.

Сетевые параметры и безопасность

Далее описаны настройки, связанные с адресацией, маршрутизацией, внешним трафиком и безопасностью сети в кластере Deckhouse Kubernetes Platform, развернутом в Yandex Cloud.

Внутренняя адресация узлов кластера

Параметр `nodeNetworkCIDR` используется для задания диапазона IP-адресов, который будет распределён между зонами доступности и применён к внутренним интерфейсам узлов.

```
nodeNetworkCIDR: 192.168.12.13/24
```

В зависимости от выбранной схемы размещения (Standard или WithNATInstance), эта подсеть будет автоматически разделена на три равные части для каждой зоны размещения:

- ru-central1-a
- ru-central1-b
- ru-central1-d

Каждая из этих частей будет использоваться как отдельная внутренняя подсеть (subnet), к которой будут подключены узлы, создаваемые в соответствующей зоне.

i В случае, если вы планируете использовать одну и ту же подсеть в нескольких кластерах (например, с `cni-simple-bridge`), необходимо учитывать ограничения: один кластер = одна таблица маршрутизации = одна подсеть. Разворачивание двух кластеров в одних и тех же подсетях с `cni-simple-bridge` невозможно. Если требуется использовать одинаковые подсети — используйте `cni-cilium`.

Назначение внешних IP-адресов и исходящего трафика

Параметр `externalSubnetIDs` указывается в секциях `masterNodeGroup.instanceClass` и `nodeGroups.instanceClass` и представляет собой массив идентификаторов подсетей Yandex Cloud, в которые будут подключены внешние сетевые интерфейсы узлов. Этот параметр обязателен, если требуется:

- Назначение публичного IP-адреса;
- Определение маршрута по умолчанию для исходящего трафика с узлов;
- Использование значений "Auto" в поле `externalIPAddresses`.

Пример:

```
externalSubnetIDs:
- <RU-CENTRAL1-A-SUBNET-ID>
- <RU-CENTRAL1-B-SUBNET-ID>
- <RU-CENTRAL1-D-SUBNET-ID>
```

i Параметр `externalSubnetIDs` обязателен для корректной работы автоматического назначения публичных IP через `externalIPAddresses: ["Auto", ...]`.

Настройка DNS и DHCP для внутренних сетей

Параметр `dhcpOptions` позволяет задать настройки DHCP-сервера, которые будут применены ко всем подсетям, создаваемым в рамках кластера Deckhouse Kubernetes Platform в Yandex Cloud.

Доступные поля:

- `domainName` — поисковый домен (search domain), который будет установлен в конфигурации сети.
- `domainNameServers` — массив IP-адресов DNS-серверов, которые будут использоваться как рекурсивные резолверы.

Пример:

```
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - 192.168.0.2
  - 192.168.0.3
```

i Указанные DNS-серверы в `domainNameServers` обязательно должны разрешать внешние и внутренние доменные зоны, используемые в кластере. В противном случае возможны сбои в работе сервисов.

После внесения изменений в `dhcpOptions`:

- Принудительно обновите `DHCP lease` на всех виртуальных машинах;
- Перезапустите все поды с `hostNetwork: true` — в частности, `kube-dns`, чтобы пересчитать содержимое `resolv.conf`.

Для применения изменений можно использовать:

```
netplan apply
```

или другой способ, соответствующий вашей системе (например, `systemd-networkd`, `dhclient` и т.д.).

Использование заранее созданных подсетей

Параметр `existingZoneToSubnetIDMap` позволяет указать соответствия между зонами доступности и ранее созданными подсетями в Yandex Cloud. Это особенно важно, если вы не хотите, чтобы DKP автоматически

создавал подсети, а хотите использовать существующие.

Пример использования:

```
existingZoneToSubnetIDMap:
  ru-central1-a: e2lu8r1tbbtryhdpa9ro
  ru-central1-b: e2lu8r1tbbtryhdpa9ro
  ru-central1-d: e2lu8r1tbbtryhdpa9ro
```

i DKP автоматически создаёт таблицу маршрутизации и не привязывает её к подсетям — это необходимо сделать вручную через интерфейс Yandex Cloud.

Дополнительные внешние сети

DKP позволяет явно указать список дополнительных внешних сетей, IP-адреса из которых будут интерпретироваться как публичные (External IP). Это задаётся в параметре `settings.additionalExternalNetworkIDs` в ресурсе `ModuleConfig`.

Эта настройка полезна, если:

- имеются внешние подсети, не указанные в `externalSubnetIDs`, но требующие учёта как внешние;
- требуется точный контроль над интерпретацией IP-адресов при настройке балансировки, маршрутизации и экспорта статуса;
- используются нестандартные схемы подключения, например через внешние NAT-шлюзы или ручное резервирование IP.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    additionalExternalNetworkIDs:
      - enp6t4sno
```

Если параметр не задан, DKP будет использовать только те подсети, что явно указаны в `YandexClusterConfiguration` (например, через `externalSubnetIDs`), чтобы определять публичность IP.

Настройка групп безопасности в Yandex Cloud

При создании облачной сети, Yandex Cloud добавляет группу безопасности по умолчанию для всех подключенных сетей, включая сеть кластера Deckhouse Kubernetes Platform. Эта группа безопасности по умолчанию содержит правила разрешающие любой входящий и исходящий трафик и применяется для всех подсетей облачной сети, если на объект (интерфейс ВМ) явно не назначена другая группа безопасности.

✗ Не удаляйте правила по умолчанию, разрешающие любой трафик, до того как закончите настройку правил группы безопасности. Это может нарушить работоспособность кластера.

Ниже приведены общие рекомендации по настройке групп безопасности. Некорректная настройка групп безопасности может сказаться на работоспособности кластера. Ознакомьтесь с особенностями работы групп безопасности в Yandex Cloud перед использованием в продуктивных средах.

1. Определите облачную сеть, в которой работает кластер Deckhouse Kubernetes Platform.

Название сети совпадает с полем `prefix` ресурса `ClusterConfiguration`. Его можно узнать с помощью команды:

```
d8 k get secrets -n kube-system d8-cluster-configuration -ojson | \
jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
```

2. В консоли Yandex Cloud выберите сервис Virtual Private Cloud и перейдите в раздел *Группы безопасности*. У вас должна отображаться одна группа безопасности с пометкой `Default`.

Группы безопасности

Фильтровать по имени или идентификатору

Имя	Идентификатор	Описание	Статус
default-sg-enpcnh1tm19i7722oap3	Default enprl7sg8f7u0eglcmls	Default security group for network	Active

3. Создайте правила согласно инструкции Yandex Cloud.

Правила

Входящий трафик Исходящий трафик

Протокол	Диапазон портов	Тип источника	Источник	Описание	
TCP	0-65535	Группа безопасности	Balancer	—	...
Any	0-65535	Группа безопасности	Self	—	...
Any	—	CIDR	10.111.0.0/16	podSubnetCIDR	...
Any	—	CIDR	10.222.0.0/16	serviceSubnetCIDR	...
ICMP	—	CIDR	10.0.0.0/8 +2	—	...
TCP	30000-32767	CIDR	0.0.0.0/0	—	...
TCP	22	CIDR	0.0.0.0/0	—	...

Добавить правило

Сохранить

4. Удалите правило, разрешающее любой **входящий** трафик (на скриншоте выше оно уже удалено), и сохраните изменения.

Настройка доступа через bastion-хост

Для подключения к узлам, находящимся в частных подсетях (например, при использовании схемы размещения Standard или WithNATInstance), используется bastion-хост — промежуточная машина с публичным IP, через которую осуществляется SSH-доступ к узлам.

Для настройки доступа выполните следующие шаги:

- 1. Выполните bootstrap базовой инфраструктуры. Перед созданием bastion-хоста необходимо выполнить начальную фазу установки DKP, которая подготовит сетевую инфраструктуру:

```
dhctl bootstrap-phase base-infra --config config.yml
```

- 2. Создайте bastion-хост в Yandex Cloud:

```
yc compute instance create \
  --name bastion \
  --hostname bastion \
  --create-boot-disk image-family=ubuntu-2204-lts,image-folder-id=standard-images,size=20,type=network-hdd \
  --memory 2 \
  --cores 2 \
  --core-fraction 100 \
  --ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
  --zone ru-central1-a \
  --public-address 178.154.226.159
```

i Замените здесь `<SSH_PUBLIC_KEY_FILE>` на имя вашего публичного ключа. Например, для ключа с RSA-шифрованием это будет `id_rsa.pub`, а для ключа с ED25519-шифрованием `id_ed25519.pub`.

Убедитесь, что IP-адрес из параметра `--public-address` доступен из вашей сети и указан корректно.

3. Запустите основной bootstrap DKP через bastion-хост:

```
dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
  --ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --config=/config.yml
```

i Замените здесь `<SSH_PRIVATE_KEY_FILE>` на имя вашего приватного ключа. Например, для ключа с RSA-шифрованием это может быть `id_rsa`, а для ключа с ED25519-шифрованием — `id_ed25519`.

Здесь:

- `--ssh-bastion-user` — пользователь для подключения к bastion-хосту;
- `--ssh-user` — пользователь на целевых узлах кластера;
- `--ssh-agent-private-keys` — путь до приватного SSH-ключа;
- `--config` — путь до конфигурационного файла DKP.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `cloud-provider-yandex`
- Настройки провайдера модуля `cloud-provider-yandex`
- Документация модуля `cni-cilium`
- Документация модуля `cni-simple-bridge`

Хранилище и балансировка нагрузки в Yandex Cloud

Этот раздел охватывает дополнительные аспекты интеграции Deckhouse Kubernetes Platform (DKP) с Yandex Cloud:

- подключение облачных дисков через CSI;
- автоматическое создание `StorageClass`;
- использование балансировщиков нагрузки;
- особенности применения изменений;
- работу с CloudStatic-узлами и bastion-хостами.

Хранилище (CSI и StorageClass)

DKP обеспечивает интеграцию с блочным хранилищем Yandex Cloud через компонент Container Storage Interface (CSI). Это даёт возможность кластерам DKP автоматически заказывать и подключать диски, а также использовать стандартные Kubernetes-ресурсы `PersistentVolumeClaim` для работы с хранилищем.

DKP автоматически создаёт ресурсы `StorageClass` для всех поддерживаемых типов дисков Yandex Cloud. Это делает возможным для всех пользователей сразу использовать хранилище, не создавая вручную описания классов.

Поддерживаются следующие типы дисков:

Тип диска	Имя <code>StorageClass</code>	Комментарии
<code>network-hdd</code>	<code>network-hdd</code>	—
<code>network-ssd</code>	<code>network-ssd</code>	—
<code>network-ssd-nonreplicated</code>	<code>network-ssd-nonreplicated</code>	Размер кратен 93 ГБ
<code>network-ssd-io-m3</code>	<code>network-ssd-io-m3</code>	Размер кратен 93 ГБ

i Размеры дисков `network-ssd-nonreplicated` и `network-ssd-io-m3` должны быть кратны 93 ГБ, иначе произойдёт ошибка при заказе тома.

Исключение ненужных StorageClass

Если в кластере не планируется использовать определённые типы дисков, можно отключить автоматическое создание соответствующих StorageClass. Это делается с помощью параметра `settings.storageClass.exclude` в ресурсе `ModuleConfig`:

```
settings:
  storageClass:
    exclude:
      - network-ssd-.*
      - network-hdd
```

В приведённом примере DKP не создаст StorageClass для всех `network-ssd` дисков и для `network-hdd`.

Назначение StorageClass по умолчанию

По умолчанию DKP выбирает StorageClass на основе аннотации `storageclass.kubernetes.io/is-default-class=true`.

Чтобы задать другой StorageClass по умолчанию, необходимо использовать глобальный параметр DKP `global.defaultClusterStorageClass`. Изменить его можно следующей командой:

```
d8 k edit mc global
```

Если параметр `defaultClusterStorageClass` не указан, платформа будет определять StorageClass, используемый по умолчанию, в следующем порядке:

- StorageClass с аннотацией `storageclass.kubernetes.io/is-default-class='true'` (если такой имеется в кластере).
- Первый StorageClass по алфавиту среди тех, что автоматически создаются облачным провайдером.
- По умолчанию значение параметра `defaultClusterStorageClass` — пустая строка (`" "`).

Изменение размера PVC

Размер существующего PVC можно увеличить путем изменения значения параметра `spec.resources.requests.storage` без остановки и пересоздания использующего его пода.

После изменения значения `spec.resources.requests.storage` CSI-драйвер последовательно:

- увеличивает размер диска в Yandex Cloud;
- обновляет размер связанного PersistentVolume;
- выполняет расширение файловой системы на узле, к которому подключён том.

Во время операции под продолжает работать, а смонтированный том остаётся доступным приложению. После завершения увеличения новый размер файловой системы становится доступен внутри контейнера без перезапуска пода.

i Уменьшение размера PVC не поддерживается.

Чтобы увеличить PVC, выполните следующие действия:

1. Получите имя StorageClass, используемого PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC> \
  -o jsonpath='{.spec.storageClassName}{"\n"}'
```

Где:

- `<НЕЙМСПЕЙС>` — неймспейс, в котором находится PVC;
- `<ИМЯ_PVC>` — имя PVC, размер которого необходимо увеличить.

Например:

```
d8 k -n production get pvc application-data \
  -o jsonpath='{.spec.storageClassName}{"\n"}'
```

Пример вывода команды:

```
network-ssd
```

В этом примере PVC `application-data` использует StorageClass `network-ssd`.

2. Убедитесь, что StorageClass разрешает увеличение томов:

```
d8 k get storageclass <ИМЯ_STORAGECLASS> \
  -o jsonpath='{.allowVolumeExpansion}{"\n"}'
```

Где `<ИМЯ_STORAGECLASS>` — имя StorageClass, полученное на предыдущем шаге.

Например:

```
d8 k get storageclass network-ssd \
-o jsonpath='{.allowVolumeExpansion}{""\n}'
```

Пример вывода команды:

```
true
```

3. Проверьте текущее состояние и размер PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production get pvc application-data
```

Пример вывода:

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	20Gi	RWO
network-ssd				

Убедитесь, что:

- PVC находится в состоянии `Bound`;
- в поле `CAPACITY` указан текущий размер PVC;
- в поле `STORAGECLASS` указан StorageClass, проверенный на предыдущем шаге.

4. Увеличьте размер PVC:

```
d8 k -n <НЕЙМСПЕЙС> edit pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production edit pvc application-data
```

В поле `spec.resources.requests.storage` укажите новый размер PVC:

```
spec:
  resources:
    requests:
      storage: 30Gi
```

В этом примере размер PVC увеличивается до 30Gi.

Сохраните изменения и закройте редактор.



Для StorageClass `network-ssd-nonreplicated` и `network-ssd-io-m3` размер должен быть кратен 93Gi.

5. Дождитесь увеличения PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC> --watch
```

Где:

- `<НЕЙМСПЕЙС>` — неймспейс, в котором находится PVC;
- `<ИМЯ_PVC>` — имя PVC, размер которого увеличивается.

Например:

```
d8 k -n production get pvc application-data --watch
```

Во время увеличения в поле `CAPACITY` может отображаться прежний размер:

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	20Gi	RWO
network-ssd				

Операция завершена, когда в поле `CAPACITY` отображается новый размер PVC:

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	30Gi	RWO
network-ssd				

6. Проверьте события PVC:

```
d8 k -n <НЕЙМСПЕЙС> describe pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production describe pvc application-data
```

Во время увеличения могут появиться следующие события:

```
ExternalExpanding
Resizing
FileSystemResizeRequired
```

Об успешном увеличении файловой системы свидетельствует событие:

```
FileSystemResizeSuccessful
```

Например:

```
Normal FileSystemResizeSuccessful kubelet MountVolume.NodeExpandVolume succeeded for volume
"pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
```

7. Проверьте размер файловой системы внутри пода:

```
d8 k -n <НЕЙМСПЕЙС> exec <ИМЯ_ПОДА> -- \
df -hT <ТОЧКА_МОНТИРОВАНИЯ>
```

Где:

- **<НЕЙМСПЕЙС>** — неймспейс, в котором находится под;
- **<ИМЯ_ПОДА>** — имя пода, использующего PVC;
- **<ТОЧКА_МОНТИРОВАНИЯ>** — путь внутри контейнера, в который смонтирован PVC.

Например:

```
d8 k -n production exec application-0 -- \
df -hT /data
```

Пример вывода:

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
/dev/vde	ext4	29.4G	22M	29.4G	1%	/data

i Фактический размер файловой системы может быть немного меньше размера PVC из-за служебных данных файловой системы.

Балансировка нагрузки

Внешний LoadBalancer

DKP автоматически подписывается на Kubernetes-объекты Service с типом LoadBalancer. При их создании в кластере, он создаёт соответствующие ресурсы:

- **NetworkLoadBalancer** — сетевой балансировщик нагрузки в Yandex Cloud;
- **TargetGroup** — группа конечных точек для балансировки трафика.

Эти ресурсы предоставляют Kubernetes-сервисам с типом LoadBalancer возможность принимать входящий трафик из интернета или внутренних сетей в зависимости от настроек.

Подробнее об архитектуре см. в документации Kubernetes Cloud Controller Manager for Yandex Cloud.

Внутренний LoadBalancer

Чтобы создать внутренний балансировщик (INTERNAL LoadBalancer), укажите подсеть, в которой должен быть создан listener:

Для этого добавьте следующую аннотацию в объект Service:

```
metadata:
  annotations:
    yandex.cpi.flant.com/listener-subnet-id: <SubnetID>
```

Значение `SubnetID` — это ID подсети, в которой будет создан внутренний слушатель Yandex LoadBalancer. Использование этой аннотации даёт возможность контролировать сетевую доступность балансировщика, ограничивая его только внутренними адресами.

i Поведение по умолчанию (внешний или внутренний LB) зависит от конфигурации кластера. Для явного выбора типа используйте аннотацию `yandex.cpi.flant.com/loadbalancer-external`.

Аннотации объекта Service

В кластере заданы значения по умолчанию для размещения ресурсов балансировщиков (сеть для целевой группы и подсеть для Listener). Эти значения выставляются автоматически во время развёртывания кластера и могут быть переопределены аннотациями на уровне конкретного Service.

В Yandex Cloud Controller Manager поддерживаются следующие аннотации:

- `yandex.cpi.flant.com/target-group-network-id` — указывает NetworkID, в котором будет создана целевая группа для данного Service. Переопределяет соответствующее значение по умолчанию.
- `yandex.cpi.flant.com/listener-subnet-id` — задаёт SubnetID для Listener'ов создаваемого LB для данного Service. Переопределяет соответствующее значение по умолчанию.
- `yandex.cpi.flant.com/listener-address-ipv4` — задаёт предопределённый IPv4-адрес для Listener'ов (поддерживаются и внутренние, и внешние LB).
- `yandex.cpi.flant.com/loadbalancer-external` — включает создание внешнего (external) LB для данного Service (используйте, если нужно явно создать внешний балансировщик). Переопределяет поведение по умолчанию.
- `yandex.cpi.flant.com/target-group-name-prefix` — задаёт префикс имени целевой группы, которую использует LoadBalancer. Аннотация на Service определяет целевую группу, но не выбирает узлы. Чтобы включить узлы в целевую группу, задайте аннотацию с тем же значением в параметре `spec.nodeTemplate.annotations` ресурса NodeGroup. Yandex Cloud Controller Manager включает в целевую группу все подходящие узлы с таким значением аннотации независимо от их NodeGroup. Имя целевой группы формируется как `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`.

Если для control plane или master-узлов создаются отдельные целевые группы, добавьте на master-узлы лейбл `node.kubernetes.io/exclude-from-external-load-balancers: ""`. Это предотвратит попытки контроллера автоматически добавлять master-узлы в новые целевые группы для балансировщиков. Если вы создаёте собственный балансировщик для master-узлов и хотите, чтобы YCC также мог размещать свои балансировщики на master-узлах, заранее создайте целевую группу с именем по маске `${CLUSTER-NAME}${VPC.ID}`.

Использование отдельной целевой группы для NodeGroup

По умолчанию Yandex Cloud Controller Manager добавляет все подходящие узлы кластера в целевую группу по умолчанию. Чтобы назначить узлам отдельную целевую группу, используйте аннотацию `yandex.cpi.flant.com/target-group-name-prefix`.

- В NodeGroup, узлы которой должны входить в отдельную целевую группу, задайте аннотацию `yandex.cpi.flant.com/target-group-name-prefix` в параметре `spec.nodeTemplate.annotations`. Например:

```
spec:
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
```

Аннотация распространяется на объекты Node. Yandex Cloud Controller Manager добавляет в целевую группу все подходящие узлы с таким значением аннотации независимо от их NodeGroup.

- В объекте Service типа LoadBalancer задайте ту же аннотацию с тем же значением. Например:

```
metadata:
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
```

Аннотация на Service определяет целевую группу, которую использует LoadBalancer, но не выбирает узлы.

- Убедитесь, что значения `yandex.cpi.flant.com/target-group-name-prefix` в NodeGroup и Service совпадают. В примере выше в обоих ресурсах используется значение `frontend-`.
- Целевая группа будет создана с именем, сформированным по шаблону:

```
<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
```

Например, если значение аннотации — `frontend-`, имя кластера — `my-cluster-`, а идентификатор сети — `enp123456789`, имя целевой группы будет следующим:

```
frontend-my-cluster-enp123456789
```

Пример конфигурации:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
  ---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer
```

⚠ Yandex Cloud не позволяет одному целевому ресурсу (target), определяемому парой (SubnetID, IP), одновременно находиться в нескольких целевых группах.

При изменении целевой группы Yandex Cloud Controller Manager сначала удаляет ресурс из текущей группы, а затем добавляет в новую. Во время перемещения возможен кратковременный перерыв в обработке трафика.

Узлы, для которых задан отдельный префикс целевой группы, исключаются из целевой группы по умолчанию. Поэтому балансировщики, использующие её, перестают направлять трафик на такие узлы.

Такое автоматическое перемещение выполняется только между целевыми группами, которыми управляет Yandex Cloud Controller Manager. Если ресурс уже находится в другой целевой группе, не управляемой Yandex Cloud Controller Manager, удалите ресурс из неё перед использованием аннотации.

После изменения или удаления префикса прежняя целевая группа может остаться пустой. Если она больше не используется балансировщиками, удалите её вручную.

Проверки состояния целевой группы

Параметры healthcheck'ов (для создаваемых целевых групп балансировщика):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — как часто запускать проверку, в секундах (по умолчанию 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — сколько ждать ответа от эндпоинта, в секундах. Если за это время ответ не получен, проверка считается неуспешной (по умолчанию 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — сколько подряд неуспешных проверок нужно, чтобы пометить эндпоинт как неработоспособный (unhealthy) и исключить его из балансировки (по умолчанию 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — сколько подряд успешных проверок нужно, чтобы вернуть эндпоинт в статус работоспособный (healthy) и снова включить его в балансировку (по умолчанию 2).

Особенности применения изменений

DKP не пересоздаёт уже существующие объекты Machine при изменении параметров. Пересоздание узлов происходит только при изменении:

- параметров в секции NodeGroup;
- параметров YandexInstanceClass.

Это поведение помогает избежать лишних операций и проста существующих узлов, однако требует ручного вмешательства при необходимости пересоздать машины.

Если вы изменили объект YandexClusterConfiguration (например, изменили параметры провайдера, схемы размещения, подсетей и т.д.), чтобы изменения вступили в силу, выполните команду:

```
dhctl converge
```

Команда иницирует пересчёт конфигурации и приводит текущее состояние кластера в соответствие с описанным в ресурсах.

Изменение `networkType` не пересоздаёт CloudEphemeral-узлы

В кластерах, использующих Machine Controller Manager (MCM), изменение параметра `networkType` в ресурсе YandexInstanceClass не приводит к автоматическому пересозданию существующих CloudEphemeral-узлов. Хотя MachineClass обновляется, тип сетевого ускорения у уже созданных виртуальных машин в Yandex Cloud не изменяется.

DKP намеренно не учитывает параметр `networkType` при определении необходимости пересоздания CloudEphemeral-узлов в MCM. Если бы он учитывался, обновление DKP привело бы к пересозданию CloudEphemeral-

узлов во всех кластерах, где `networkType` уже указан, даже если его значение не изменялось.

В Cluster API параметр `networkType` учитывается при определении необходимости пересоздания узлов, поэтому его изменение запускает их обновление автоматически. Новые CloudEphemeral NodeGroup в Yandex по умолчанию используют CAPI.

Если вы изменили `networkType` в кластере на MCM и хотите, чтобы новое значение применилось к уже существующим CloudEphemeral-узлам, пересоздайте их вручную — инструкция в документации.

Интеграция вручную созданных VM

DKP позволяет подключать существующие виртуальные машины в Yandex Cloud к Kubernetes-кластеру в качестве узлов. Такие узлы называются CloudStatic, поскольку они не управляются напрямую модулем `node-manager`, но могут использоваться в составе кластера.

Чтобы вручную подключить виртуальную машину в качестве CloudStatic-узла, необходимо:

1. Узнать актуальное значение `nodeNetworkCIDR` из кластера:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration -o json | \
jq --raw-output '.data."cloud-provider-cluster-configuration.yaml"' | base64 -d | grep
'^nodeNetworkCIDR'
```

Результатом будет строка вида:

```
nodeNetworkCIDR: 192.168.12.13/24
```

Это значение необходимо скопировать и указать как `value` в метаданных виртуальной машины.

2. Задать параметр `node-network-cidr` в метаданных VM:

```
key: node-network-cidr
value: <nodeNetworkCIDR из кластера>
```

Параметр `node-network-cidr` должен совпадать с тем значением, которое указано в объекте `YandexClusterConfiguration`, поле `nodeNetworkCIDR`.

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `cloud-provider-yandex`
- Документация модуля `node-manager`

Интеграция со службами Yandex Cloud

Deckhouse Kubernetes Platform (DKP) поддерживает нативную интеграцию с несколькими облачными сервисами Yandex Cloud. Это позволяет подключать внешний мониторинг, безопасно работать с секретами и автоматизировать синхронизацию данных между инфраструктурой и кластером.

В данном разделе описаны шаги по настройке следующих интеграций:

- с Yandex Lockbox через External Secrets Operator;
- с Yandex Managed Service for Prometheus.

Интеграция с Yandex Lockbox

DKP поддерживает интеграцию с Yandex Lockbox с помощью оператора External Secrets Operator (ESO). Это позволяет автоматически синхронизировать секреты из Lockbox с секретами Kubernetes в кластере.

Для интеграции выполните следующие шаги:

1. Создайте сервисный аккаунт, необходимый для работы External Secrets Operator:

```
yc iam service-account create --name eso-service-account
```

2. Создайте авторизованный ключ для сервисного аккаунта и сохраните его в файл:

```
yc iam key create --service-account-name eso-service-account --output authorized-key.json
```

3. Назначьте сервисному аккаунту роли `lockbox.editor`, `lockbox.payloadViewer` и `kms.keys.encrypterDecrypter` для доступа ко всем секретам каталога:

```
folder_id=<идентификатор каталога>
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.editor
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.payloadViewer
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role kms.keys.encrypterDecrypter
```

Для более тонкой настройки ознакомьтесь с управлением доступом в Yandex Lockbox.

- Установите External Secrets Operator с помощью Helm-чарта согласно инструкции.

Обратите внимание, что вам может понадобиться задать `nodeSelector`, `tolerations` и другие параметры. Для этого используйте файл `./external-secrets/values.yaml` после распаковки Helm-чарта.

Скачайте и распакуйте чарт:

```
helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
  --version 0.5.5 \
  --untar
```

Установите Helm-чарт:

```
helm install -n external-secrets --create-namespace \
  --set-file auth.json=authorized-key.json \
  external-secrets ./external-secrets/
```

Где:

- `authorized-key.json` — название файла с авторизованным ключом из шага 2.

- Создайте хранилище секретов SecretStore, содержащее секрет `sa-creds`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: SecretStore
metadata:
  name: secret-store
spec:
  provider:
    yandexlockbox:
      auth:
        authorizedKeySecretRef:
          name: sa-creds
          key: key
EOF
```

Где:

- `sa-creds` — название Secret, содержащего авторизованный ключ. Этот секрет должен появиться после установки Helm-чарта.
- `key` — название ключа в поле `.data` секрета выше.

- Проверьте статус External Secrets Operator:

```
d8 k -n external-secrets get po
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
external-secrets-55f78c44cf-dbf6q	1/1	Running	0	77m
external-secrets-cert-controller-78cbc7d9c8-rszhx	1/1	Running	0	77m
external-secrets-webhook-6d7b66758-s7v9c	1/1	Running	0	77m

Проверьте статус созданного хранилища секретов:

```
d8 k -n external-secrets get secretstores.external-secrets.io
```

Пример вывода:

NAME	AGE	STATUS
secret-store	69m	Valid

- Создайте секрет Yandex Lockbox со следующими параметрами:

- Имя — `lockbox-secret`.
- Ключ — введите неконфиденциальный идентификатор `password`.
- Значение — введите конфиденциальные данные для хранения `p@$$w0rd`.

- Создайте объект ExternalSecret, указывающий на секрет `lockbox-secret` в хранилище `secret-store`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
    - secretKey: password
      remoteRef:
        key: <ИДЕНТИФИКАТОР_СЕКРЕТА>
        property: password
EOF
```

Где:

- `spec.target.name` — имя нового секрета. External Secret Operator создаст этот секрет в кластере Deckhouse Kubernetes Platform и поместит в него параметры секрета Yandex Lockbox `lockbox-secret`.
- `spec.data[].secretKey` — название ключа в поле `.data` секрета, который создаст External Secret Operator.
- `spec.data[].remoteRef.key` — идентификатор созданного ранее секрета Yandex Lockbox `lockbox-secret`. Например, `e6q28nvmhu539*****`.
- `spec.data[].remoteRef.property` — ключ, указанный ранее, для секрета Yandex Lockbox `lockbox-secret`.

9. Убедитесь, что новый ключ `k8s-secret` содержит значение секрета `lockbox-secret`:

```
d8 k -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

В выводе команды будет содержаться значение ключа `password` секрета `lockbox-secret`, созданного ранее:

```
p@$w0rd
```

Интеграция с Yandex Managed Service for Prometheus

С помощью данной интеграции вы можете использовать Yandex Managed Service for Prometheus в качестве внешнего хранилища метрик, например, для долгосрочного хранения.

Для записи метрик (PrometheusRemoteWrite) выполните следующие шаги:

1. Создайте сервисный аккаунт с ролью `monitoring.editor`.
2. Создайте API-ключ для сервисного аккаунта.
3. Создайте ресурс PrometheusRemoteWrite:

```
d8 k apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <URL_ЗАПИСИ_МЕТРИК>
  bearerToken: <API_КЛЮЧ>
EOF
```

Где:

- `<URL_ЗАПИСИ_МЕТРИК>` — URL со страницы Yandex Monitoring/Prometheus/Запись метрик.
- `<API_КЛЮЧ>` — API-ключ, созданный на предыдущем шаге. Например, `AQVN1nHJReSrfo9jU3aopsXrJyfq_UHs*****`.

Также вы можете указать дополнительные параметры в соответствии с документацией.

Подробнее с данной функциональностью можно ознакомиться в документации Yandex Cloud.

Для чтения метрик через Grafana:

1. Создайте сервисный аккаунт с ролью `monitoring.viewer`.
2. Создайте API-ключ для сервисного аккаунта.
3. Создайте ресурс GrafanaAdditionalDatasource:

```

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <URL_ЧТЕНИЕ_МЕТРИК_ЧЕРЕЗ_GRAFANA>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_КЛЮЧ>

```

Где:

- `<URL_ЧТЕНИЕ_МЕТРИК_ЧЕРЕЗ_GRAFANA>` — URL со страницы Yandex Monitoring/Prometheus/Чтение метрик через Grafana.
- `<API_КЛЮЧ>` — API-ключ, созданный на предыдущем шаге. Например, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs*****`.

Также вы можете указать дополнительные параметры в соответствии с документацией.

Подробнее с данной функциональностью можно ознакомиться в документации Yandex Cloud.

Дополнительные ресурсы

- Кастомные ресурсы модуля `prometheus`

Подключение и авторизация в VK Cloud

Для управления ресурсами в VK Cloud с помощью Deckhouse Kubernetes Platform необходимо подключиться к VK Cloud API.

Перечень API-сервисов VK Cloud, доступ к которым необходим для развертывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к VK Cloud API, находятся в `openrc`-файле (OpenStack RC file).

Информация о получении `openrc`-файла с помощью стандартного веб-интерфейса VK Cloud и о способах его использования доступна в документации VK Cloud.

Если вы используете VK Cloud API cloud-провайдера, интерфейс получения `openrc`-файла может быть другим.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка VK Cloud

Схемы размещения

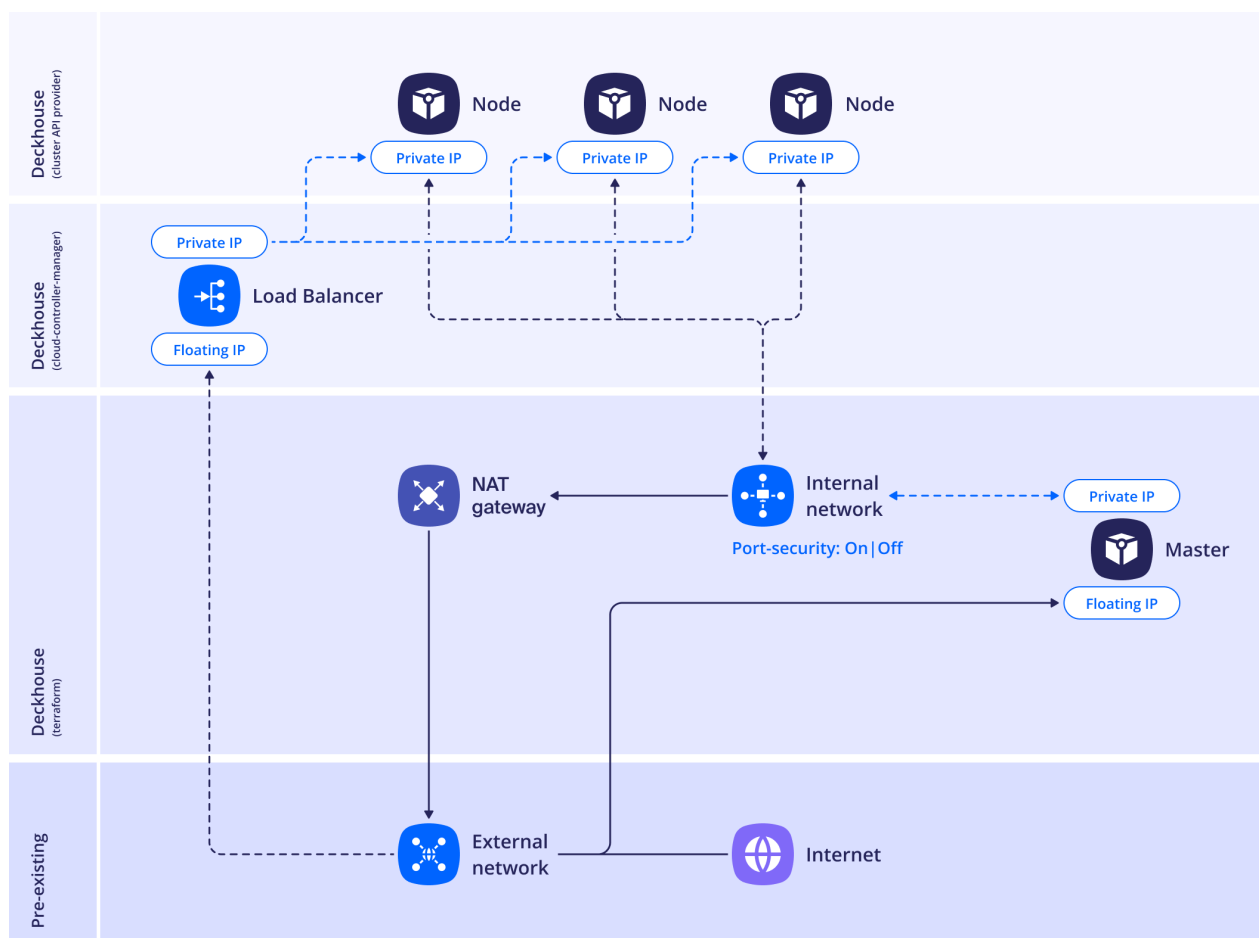
Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре VK Cloud и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.



Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, `kube-apiserver` на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                 # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false        # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared                # Обязательный параметр.
  bastion:
    zone: ru2-b                             # Необязательный параметр.
    volumeType: fast-ru-2b                   # Необязательный параметр.
    instanceClass:
      flavorName: m1.large                   # Обязательный параметр.
      imageName: ubuntu-20-04-cloud-amd64   # Обязательный параметр.
      rootDiskSize: 50                       # Обязательный параметр, по умолчанию 50 ГБ.
      additionalTags:
        severity: critical                   # Необязательный параметр.
        environment: production              # Необязательный параметр.
  masterNodeGroup:
    replicas: 3
    instanceClass:
      flavorName: m1.large                   # Обязательный параметр.
      imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
      # Необязательный параметр. Если не указан – используется локальный диск.
      rootDiskSize: 50
      # Необязательный параметр, дополнительные группы безопасности.
      additionalSecurityGroups:
        - sec_group_1
        - sec_group_2
      additionalTags:
        severity: critical
        environment: production
      # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
      # (всегда используйте самый быстрый диск, предоставленный поставщиком).
      volumeTypeMap:

```



```

# Если указан rootDiskSize, этот тип тома также будет
# использоваться для главного корневого тома.
ru-1a: fast-ru-1a
ru-1b: fast-ru-1b
ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIP Pools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...

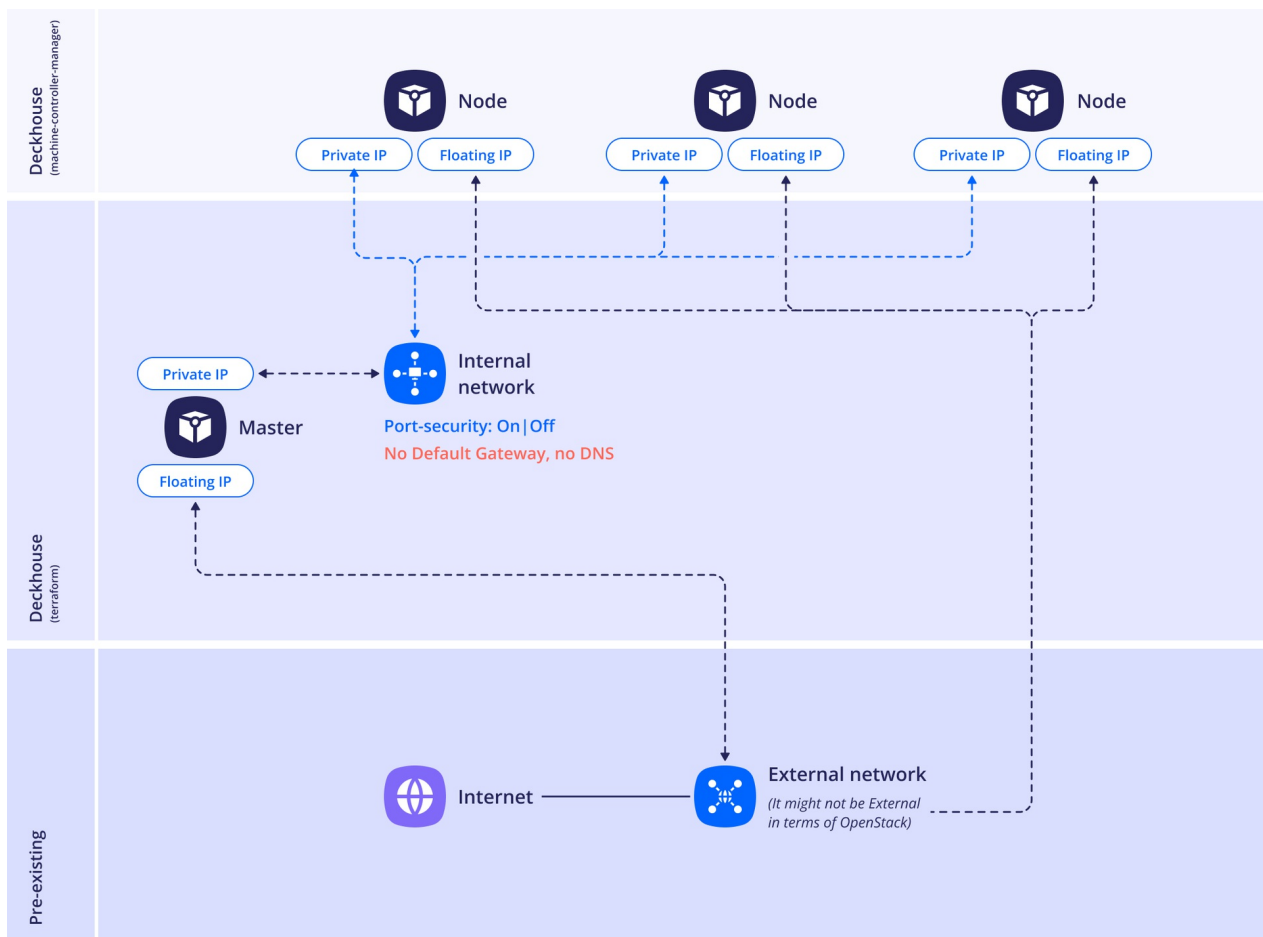
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                   # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

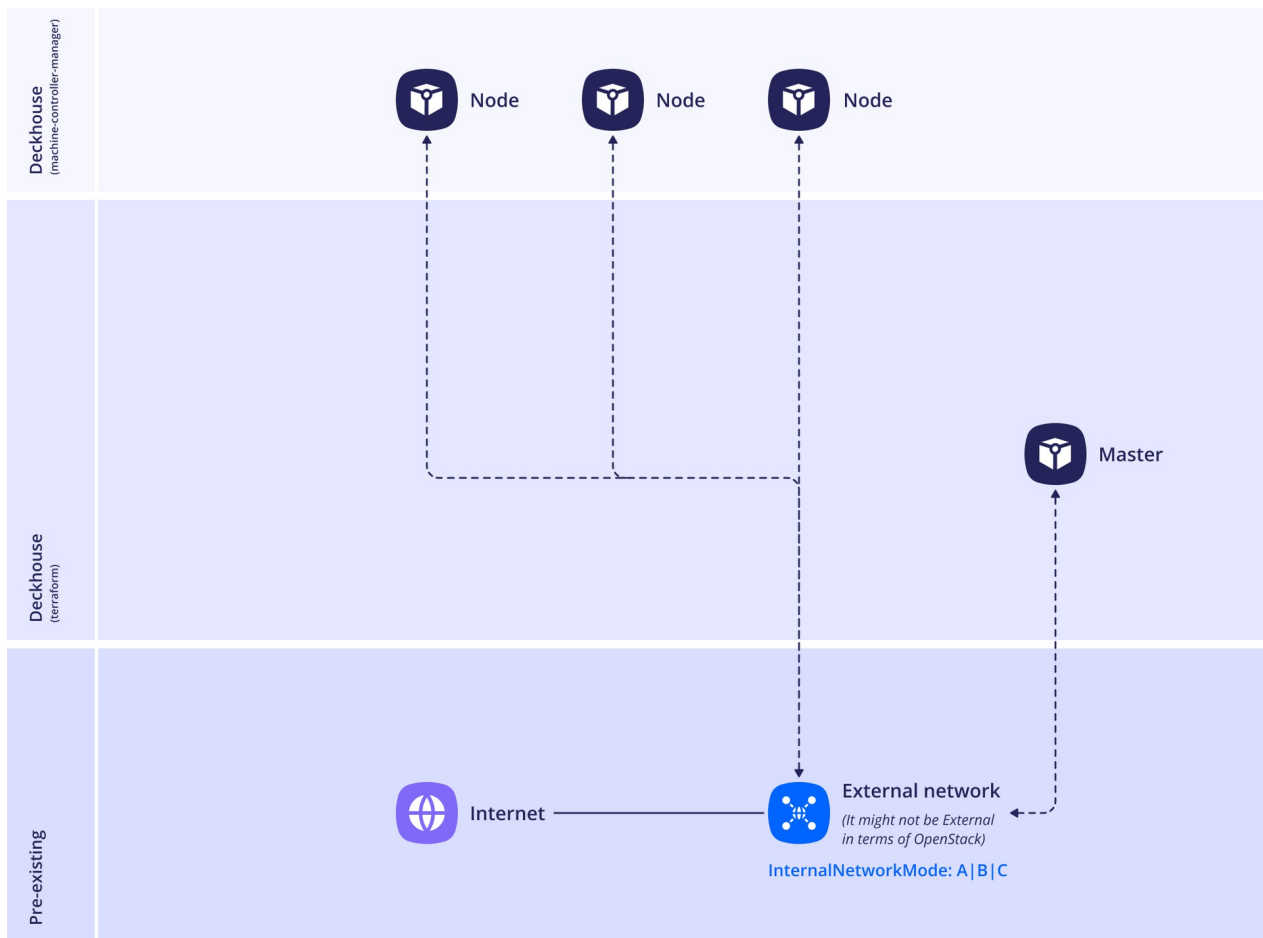
```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

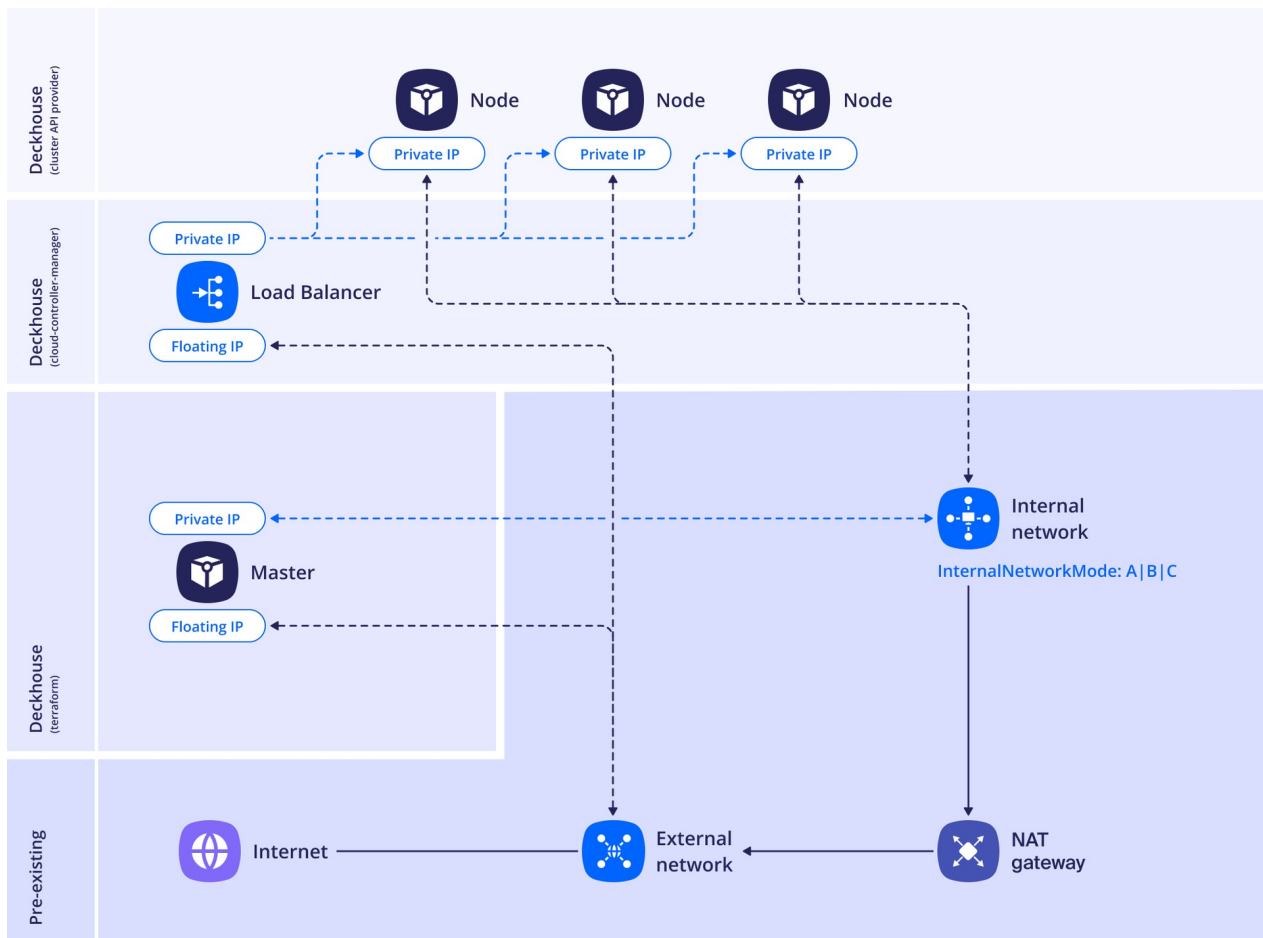
```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграции с VK Cloud осуществляется с помощью ресурса `OpenStackClusterConfiguration`, который описывает конфигурацию облачного кластера в VK Cloud и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера VK Cloud — это `custom resource OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».

 При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов VK Cloud, необходимых для работы Deckhouse Kubernetes Platform в VK Cloud:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Адреса и порты API можно узнать в официальной документации.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим VM в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в VK Cloud

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте `openrc`-файл, который содержит `credentials` для обращения к API VK Cloud.

i Интерфейс получения `openrc`-файла может отличаться в зависимости от провайдера VK Cloud. Если провайдер предоставляет стандартный интерфейс для VK Cloud, скачать `openrc`-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него `openrc`-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из `openrc`-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

ID	Name	Is Public
8d39c9db-0293-48c0-8d44-015a2f6788ff	ko1-high-iops	True
bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd	dp1-high-iops	True
74101409-a462-4f03-872a-7de727a178b8	ko1-ssd	True
eadd8860-f5a4-45e1-ae27-8c58094257e0	dp1-ssd	True
48372c05-c842-4f6e-89ca-09af3868b2c4	ssd	True
a75c3502-4de6-4876-a457-a6c4594c067a	ms1	True
ebf5922e-42af-4f97-8f23-716340290de2	dp1	True
a6e853c1-78ad-4c18-93f9-2bba317a1d13	ceph	True

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если VK Cloud не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:

```

+-----+-----+
|
|
|
+-----+-----+
| Field          | Value
|
+-----+-----+
|
+-----+-----+
| checksum       | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at      | 2020-01-10T07:23:48Z
|
| disk_format     | qcow2
|
| file            | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id              | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk        | 0
|
| min_ram         | 0
|
| name            | ubuntu-18-04-cloud-amd64
|
| owner           | bb506e3ece54e21b2acf1bf9db4f62c
|
| properties      | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected       | False
|
| schema          | /v2/schemas/image
|
| size            | 343277568
|
| status          | active
|
| tags            |
|
| updated_at      | 2020-05-01T17:18:34Z
|
| virtual_size    | None
|
| visibility      | private
|
+-----+-----+
|
|
|
+-----+-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internalurl_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internalurl }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в VK Cloud

Диск узла может быть локальным или сетевым. В терминологии VK Cloud локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в VK Cloud flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

i При создании узлов с типом CloudEphemeral в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -c disk
```

Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ VK Cloud и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Подключение и авторизация в Selectel Cloud

Для управления ресурсами в Selectel с помощью Deckhouse Kubernetes Platform необходимо подключиться к Selectel API.

Перечень API-сервисов Selectel, доступ к которым необходим для развертывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к Selectel API, находятся в openrc-файле (OpenStack RC file).

Информация о получении openrc-файла с помощью стандартного веб-интерфейса Selectel и о способах его использования доступна в документации Selectel.

Если вы используете Selectel API cloud-провайдера, интерфейс получения openrc-файла может быть другим.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка Selectel Cloud

Схемы размещения

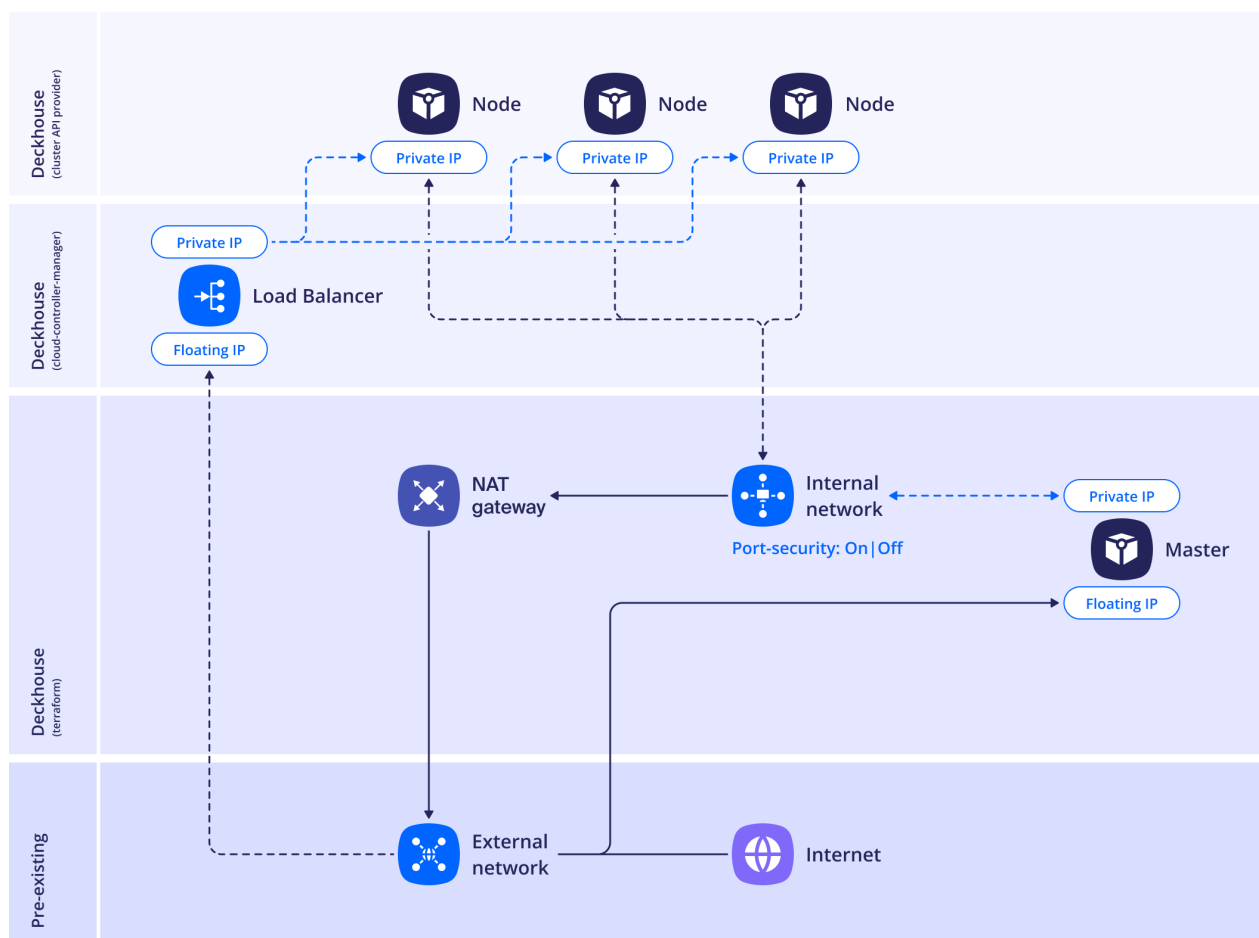
Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре Selectel и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.



Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false       # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared              # Обязательный параметр.
  bastion:
    zone: ru2-b                            # Необязательный параметр.
    volumeType: fast-ru-2b                 # Необязательный параметр.
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-20-04-cloud-amd64  # Обязательный параметр.
      rootDiskSize: 50                     # Обязательный параметр, по умолчанию 50 ГБ.
      additionalTags:
        severity: critical                 # Необязательный параметр.
        environment: production           # Необязательный параметр.
  masterNodeGroup:
    replicas: 3
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-18-04-cloud-amd64  # Обязательный параметр.
      # Необязательный параметр. Если не указан – используется локальный диск.
      rootDiskSize: 50
      # Необязательный параметр, дополнительные группы безопасности.
      additionalSecurityGroups:
        - sec_group_1
        - sec_group_2
      additionalTags:
        severity: critical
        environment: production
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставленный поставщиком).
  volumeTypeMap:

```

```

# Если указан rootDiskSize, этот тип тома также будет
# использоваться для главного корневого тома.
ru-1a: fast-ru-1a
ru-1b: fast-ru-1b
ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...

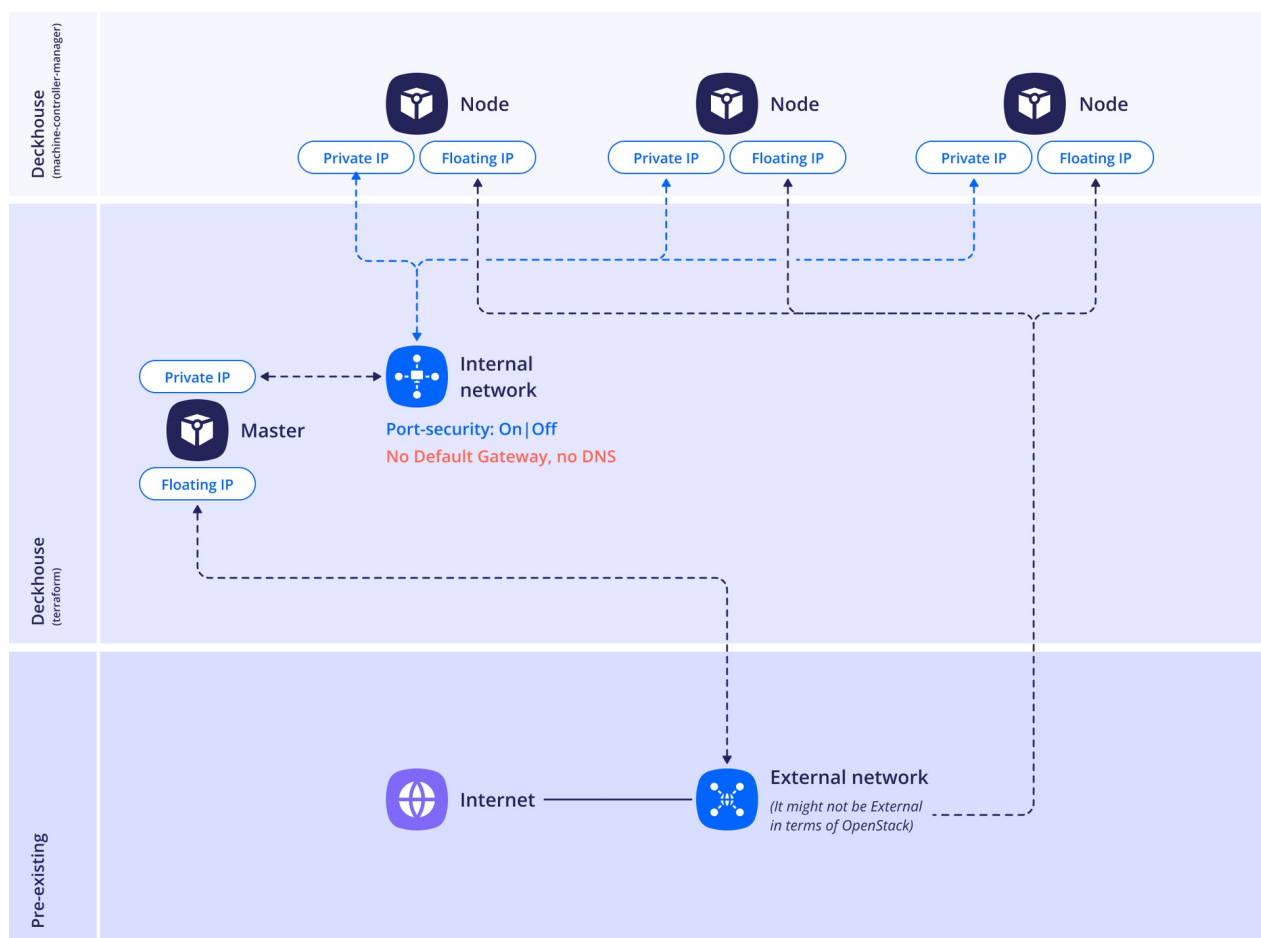
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:


```

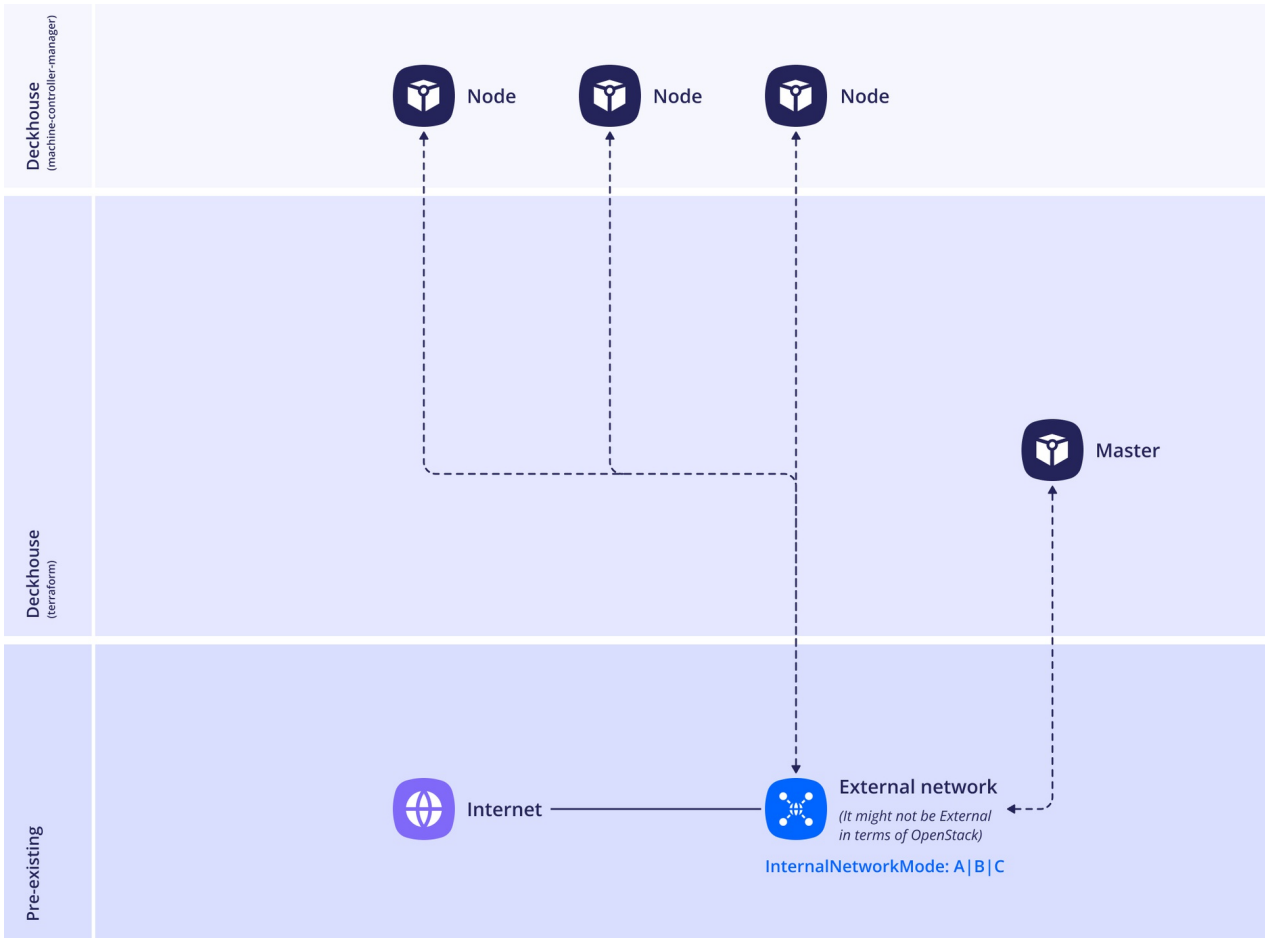
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                    # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

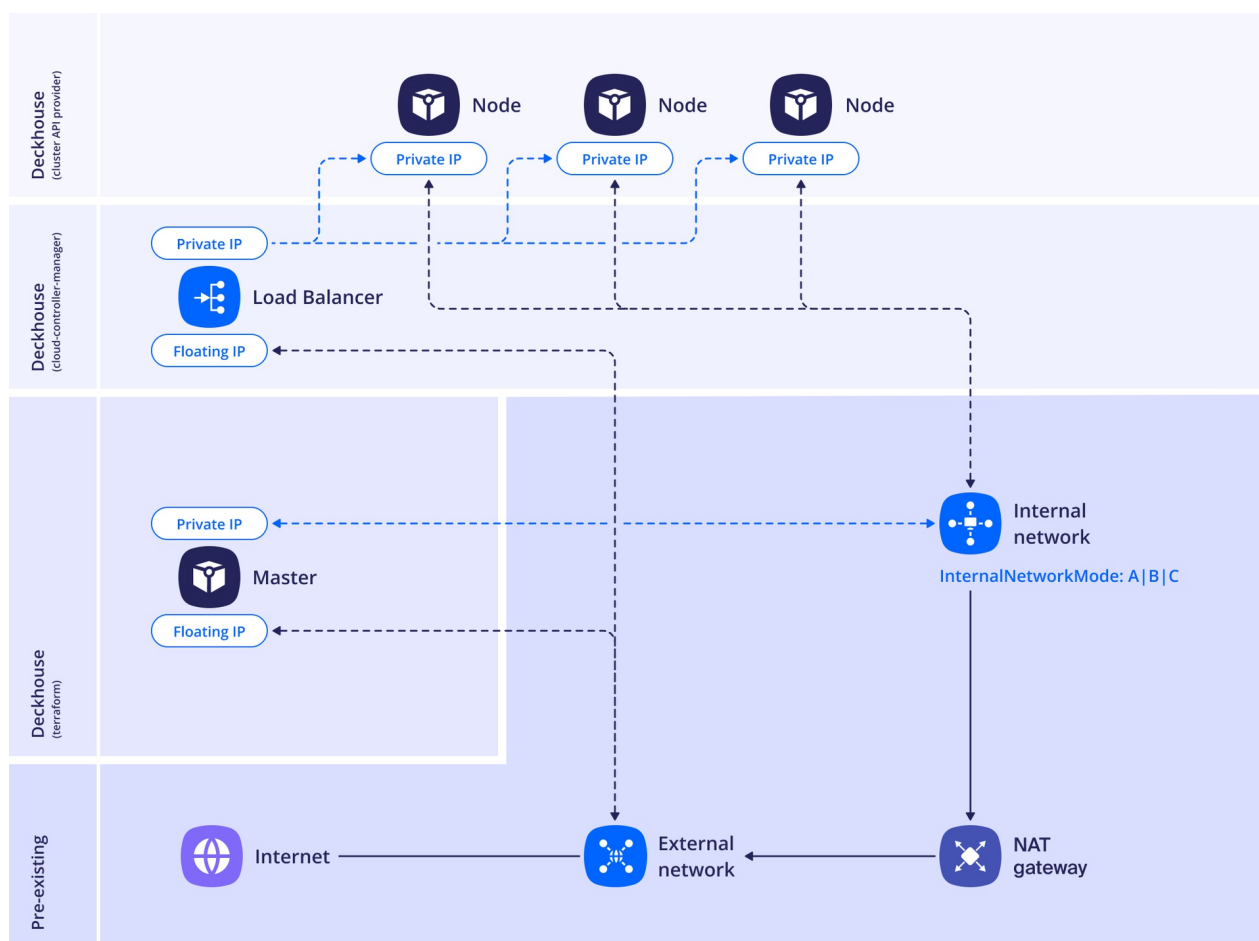
```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграция с Selectel осуществляется с помощью ресурса `OpenStackClusterConfiguration`, который описывает конфигурацию облачного кластера в Selectel и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера Selectel — это custom resource `OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».



При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов Selectel, необходимых для работы Deckhouse Kubernetes Platform в Selectel:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим ВМ в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в Selectel

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте openrc-файл, который содержит credentials для обращения к API Selectel.

i Интерфейс получения openrc-файла может отличаться в зависимости от провайдера Selectel. Если провайдер предоставляет стандартный интерфейс для Selectel, скачать openrc-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него openrc-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из openrc-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

ID	Name	Is Public
8d39c9db-0293-48c0-8d44-015a2f6788ff	ko1-high-iops	True
bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd	dp1-high-iops	True
74101409-a462-4f03-872a-7de727a178b8	ko1-ssd	True
eadd8860-f5a4-45e1-ae27-8c58094257e0	dp1-ssd	True
48372c05-c842-4f6e-89ca-09af3868b2c4	ssd	True
a75c3502-4de6-4876-a457-a6c4594c067a	ms1	True
ebf5922e-42af-4f97-8f23-716340290de2	dp1	True
a6e853c1-78ad-4c18-93f9-2bba317a1d13	ceph	True

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если Selectel не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:


```

+-----+-----+
|
|
|
+-----+-----+
| Field          | Value
|
+-----+-----+
|
+-----+-----+
| checksum        | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bb506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility        | private
|
+-----+-----+
|
|
|
+-----+-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в Selectel

Диск узла может быть локальным или сетевым. В терминологии Selectel локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в Selectel flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -C disk
```

Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ Selectel и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Оффлайн-изменение размера диска

Если при изменении размера диска вы получаете следующую ошибку, необходимо уменьшить количество реплик StatefulSet до 0, подождать изменения размера дисков и вернуть обратно количество реплик, которое было до начала операции.

```
Warning VolumeResizeFailed 5s (x11 over 41s) external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Подключение и авторизация в OVH Cloud

Для управления ресурсами в OVHcloud с помощью Deckhouse Kubernetes Platform необходимо подключиться к OVHcloud API.

Перечень API-сервисов OVHcloud, доступ к которым необходим для развертывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к OVHcloud API, находятся в openrc-файле (OpenStack RC file).

Информация о получении openrc-файла с помощью стандартного веб-интерфейса OVHcloud и о способах его использования доступна в документации OVHcloud.

Если вы используете OVHcloud API cloud-провайдера, интерфейс получения openrc-файла может быть другим.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка OVH Cloud

Схемы размещения

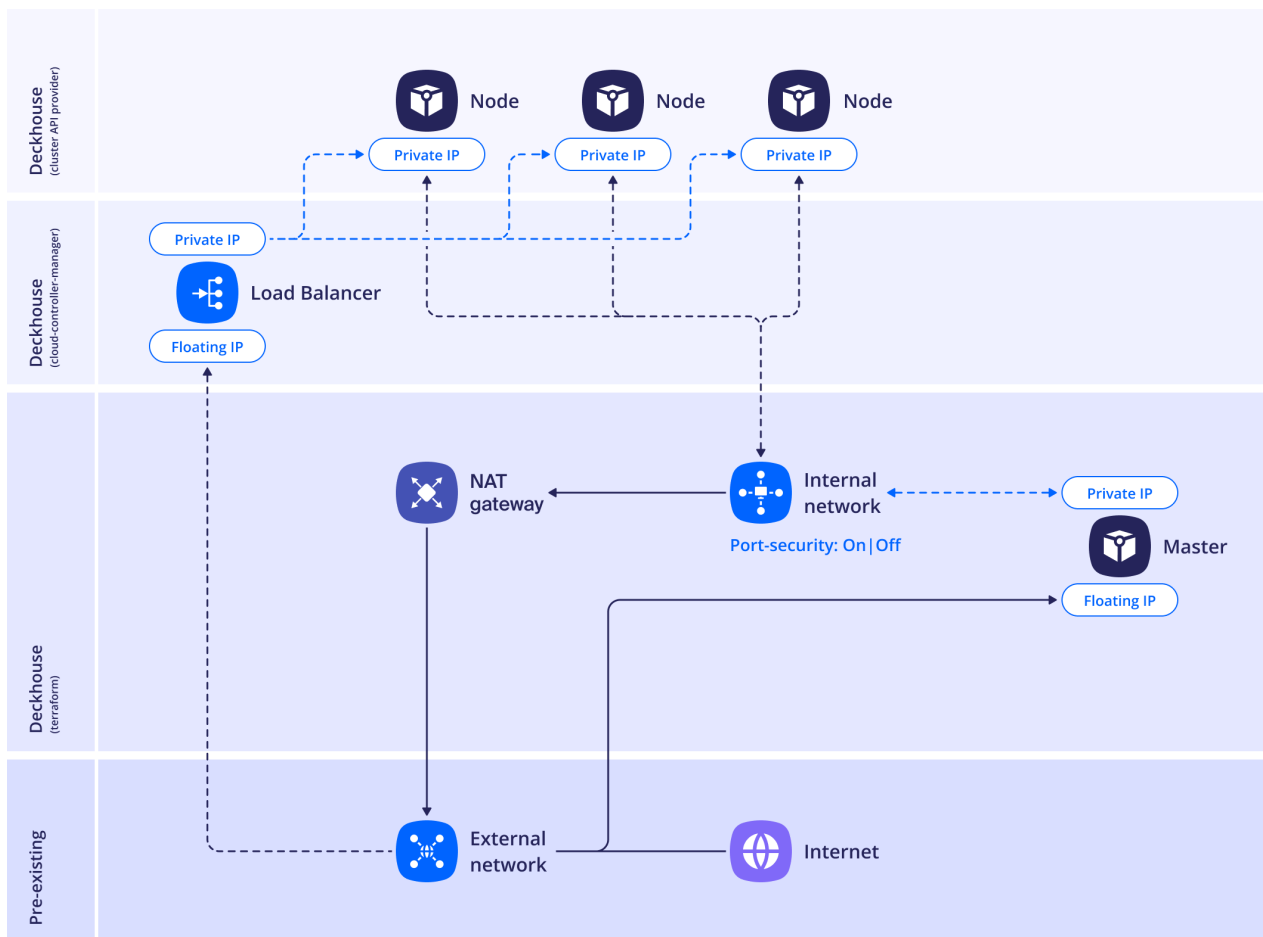
Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре OVHcloud и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.



Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false       # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared              # Обязательный параметр.
  bastion:
    zone: ru2-b                            # Необязательный параметр.
    volumeType: fast-ru-2b                 # Необязательный параметр.
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-20-04-cloud-amd64  # Обязательный параметр.
      rootDiskSize: 50                     # Обязательный параметр, по умолчанию 50 ГБ.
      additionalTags:
        severity: critical                 # Необязательный параметр.
        environment: production           # Необязательный параметр.
  masterNodeGroup:
    replicas: 3
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-18-04-cloud-amd64  # Обязательный параметр.
      # Необязательный параметр. Если не указан – используется локальный диск.
      rootDiskSize: 50
      # Необязательный параметр, дополнительные группы безопасности.
      additionalSecurityGroups:
        - sec_group_1
        - sec_group_2
      additionalTags:
        severity: critical
        environment: production
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставленный поставщиком).
  volumeTypeMap:

```

```

# Если указан rootDiskSize, этот тип тома также будет
# использоваться для главного корневого тома.
ru-1a: fast-ru-1a
ru-1b: fast-ru-1b
ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIP Pools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
  sshPublicKey: "<SSH_PUBLIC_KEY>"
  tags:
    project: cms
    owner: default
  provider:
    ...

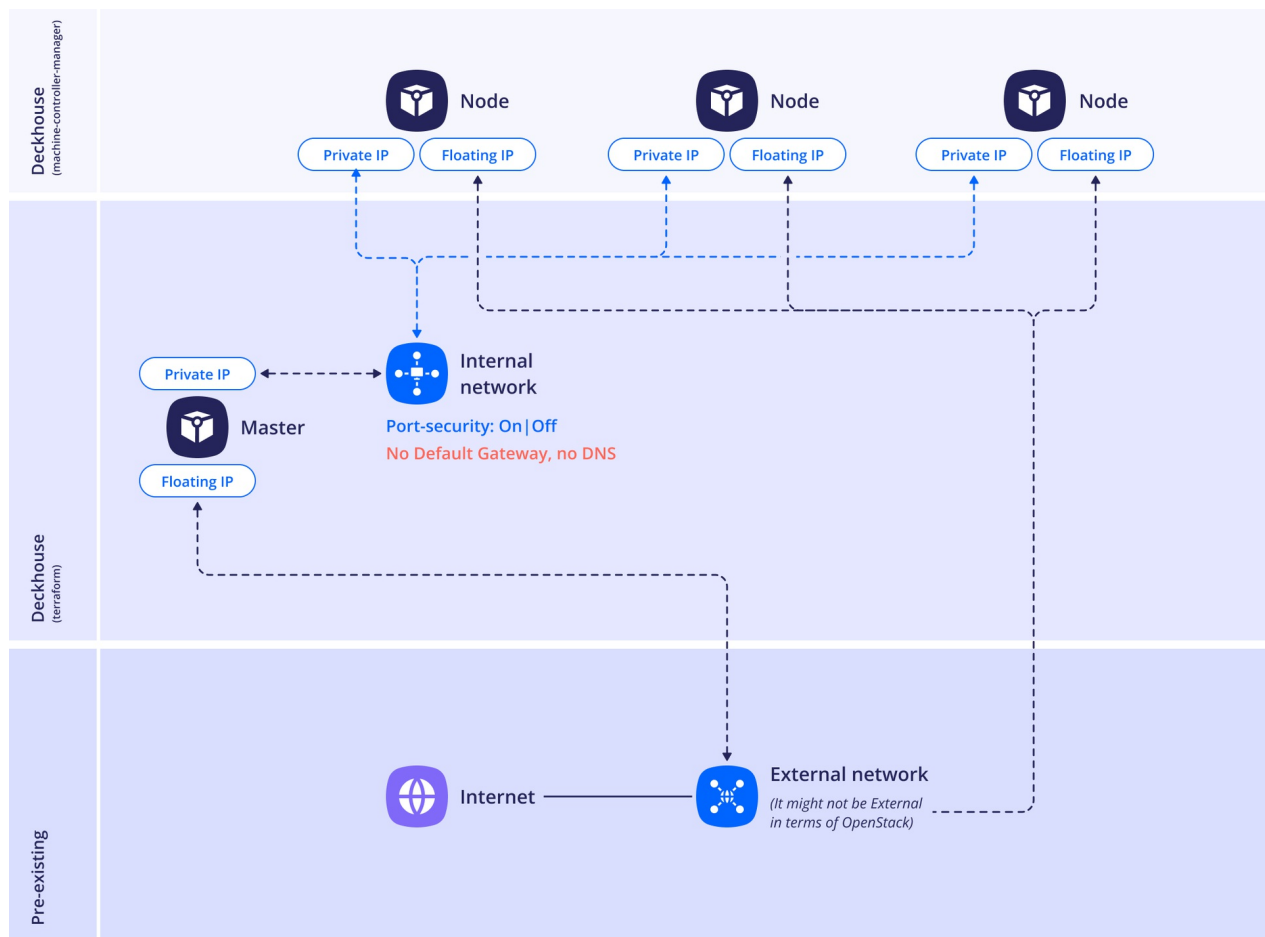
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

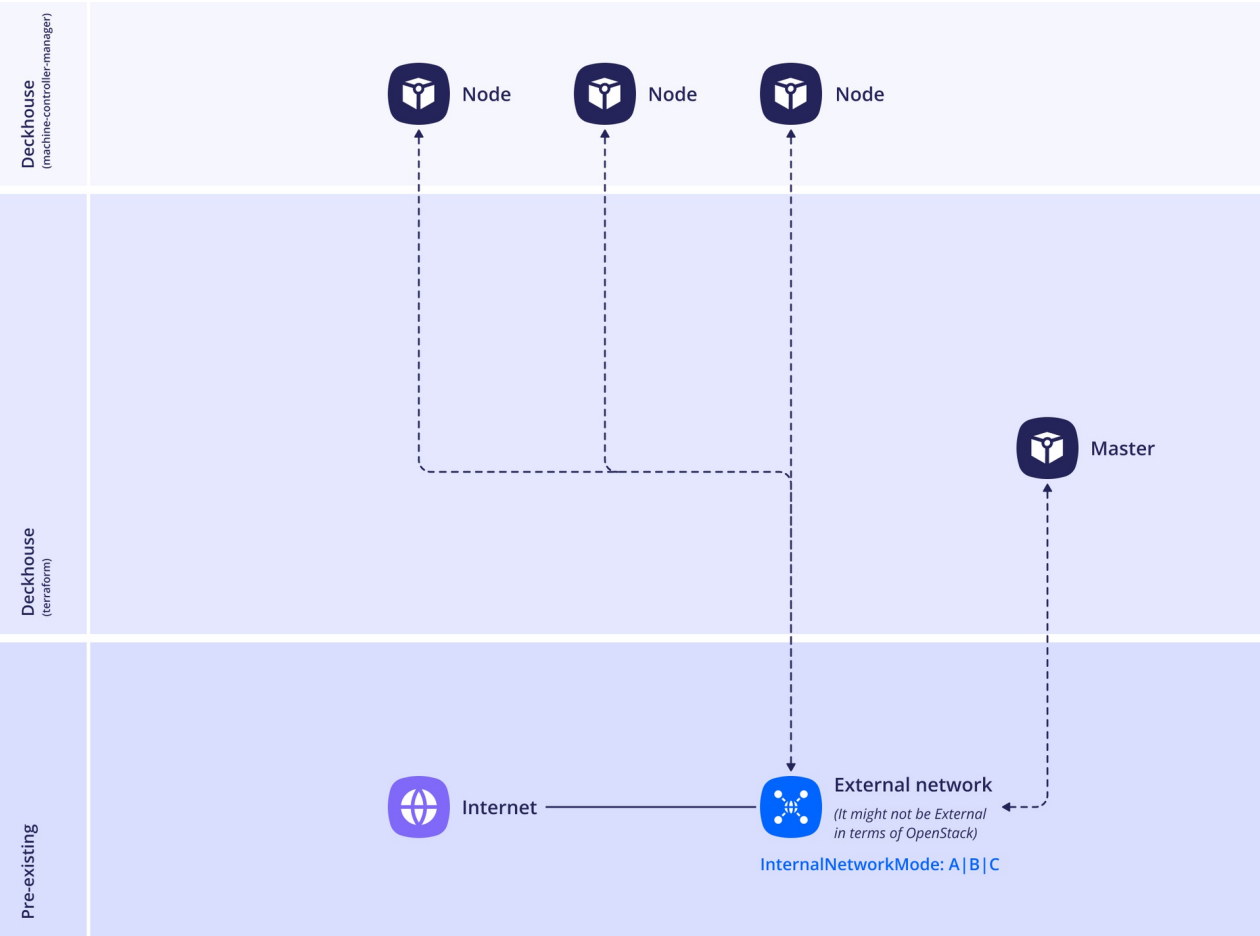
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                    # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:


```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

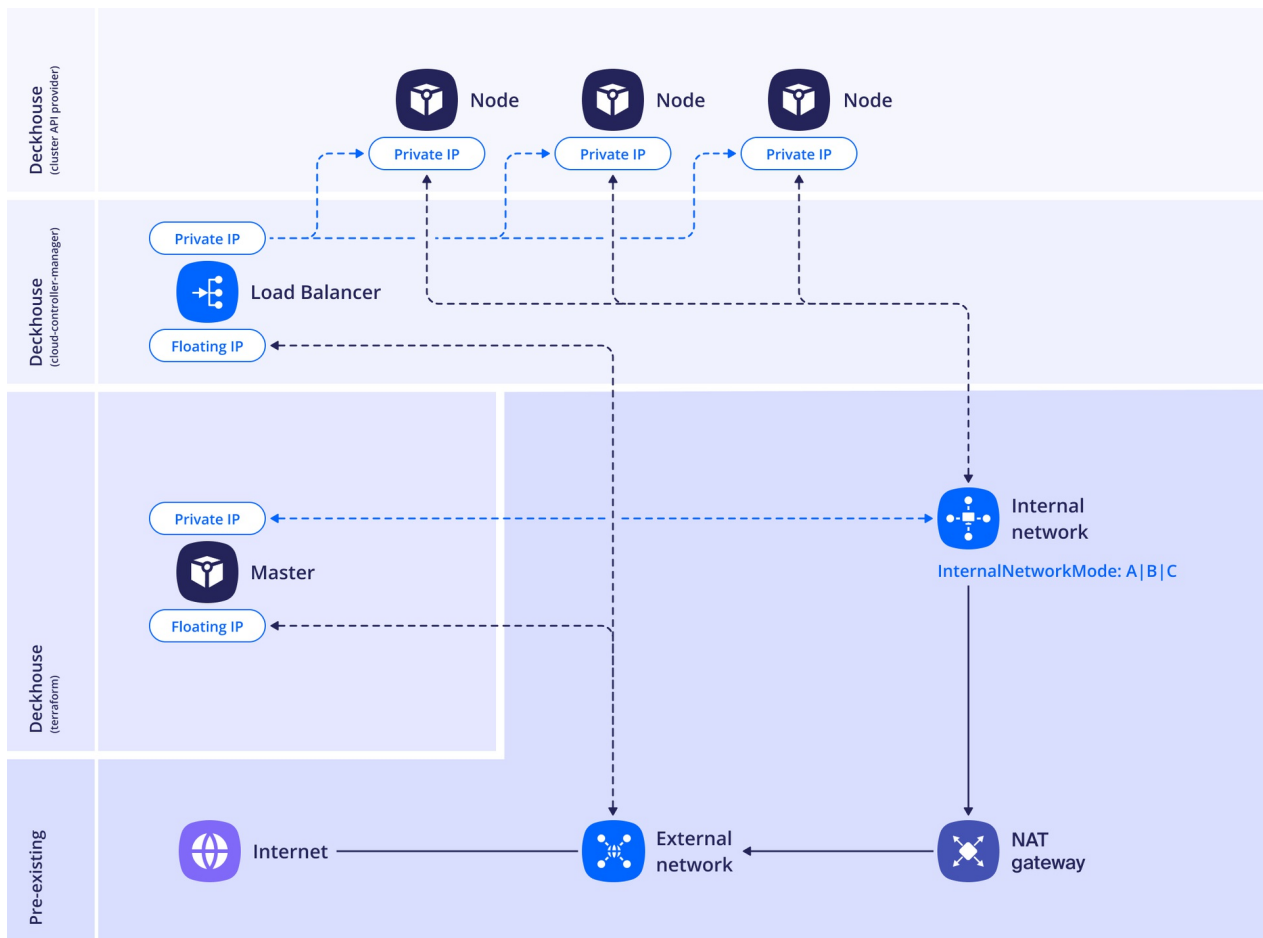
```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграция с OVHcloud осуществляется с помощью ресурса OpenStackClusterConfiguration, который описывает конфигурацию облачного кластера в OVHcloud и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера OVHcloud — это custom resource `OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».



При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов OVHcloud, необходимых для работы Deckhouse Kubernetes Platform в OVHcloud:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим ВМ в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в OVHcloud

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте `openrc`-файл, который содержит `credentials` для обращения к API OVHcloud.

i Интерфейс получения `openrc`-файла может отличаться в зависимости от провайдера OVHcloud. Если провайдер предоставляет стандартный интерфейс для OVHcloud, скачать `openrc`-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него `openrc`-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из `openrc`-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

ID	Name	Is Public
8d39c9db-0293-48c0-8d44-015a2f6788ff	ko1-high-iops	True
bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd	dp1-high-iops	True
74101409-a462-4f03-872a-7de727a178b8	ko1-ssd	True
eadd8860-f5a4-45e1-ae27-8c58094257e0	dp1-ssd	True
48372c05-c842-4f6e-89ca-09af3868b2c4	ssd	True
a75c3502-4de6-4876-a457-a6c4594c067a	ms1	True
ebf5922e-42af-4f97-8f23-716340290de2	dp1	True
a6e853c1-78ad-4c18-93f9-2bba317a1d13	ceph	True

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если OVHcloud не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:

```

+-----+-----+
|
|
|
+-----+-----+
|
|
+-----+-----+
|
|
+-----+-----+
| Field          | Value
|
+-----+-----+
|
|
+-----+-----+
| checksum        | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bb506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+-----+-----+
|
|
+-----+-----+
|
|
+-----+-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internalurl_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internalurl }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в OVHcloud

Диск узла может быть локальным или сетевым. В терминологии OVHcloud локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в OVHcloud flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

i При создании узлов с типом CloudEphemeral в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -c disk
```


Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ OVHcloud и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Офлайн-изменение размера диска

Если при изменении размера диска вы получаете следующую ошибку, необходимо уменьшить количество реплик StatefulSet до 0, подождать изменения размера дисков и вернуть обратно количество реплик, которое было до начала операции.

```
Warning VolumeResizeFailed 5s (x11 over 41s) external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Подключение и авторизация в OpenStack

Для управления ресурсами в OpenStack с помощью Deckhouse Kubernetes Platform необходимо подключиться к OpenStack API.

Перечень API-сервисов OpenStack, доступ к которым необходим для развертывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к OpenStack API, находятся в openrc-файле (OpenStack RC file).

Информация о получении openrc-файла с помощью стандартного веб-интерфейса OpenStack и о способах его использования доступна в документации OpenStack.

Если вы используете OpenStack API cloud-провайдера, интерфейс получения openrc-файла может быть другим.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка OpenStack

Схемы размещения

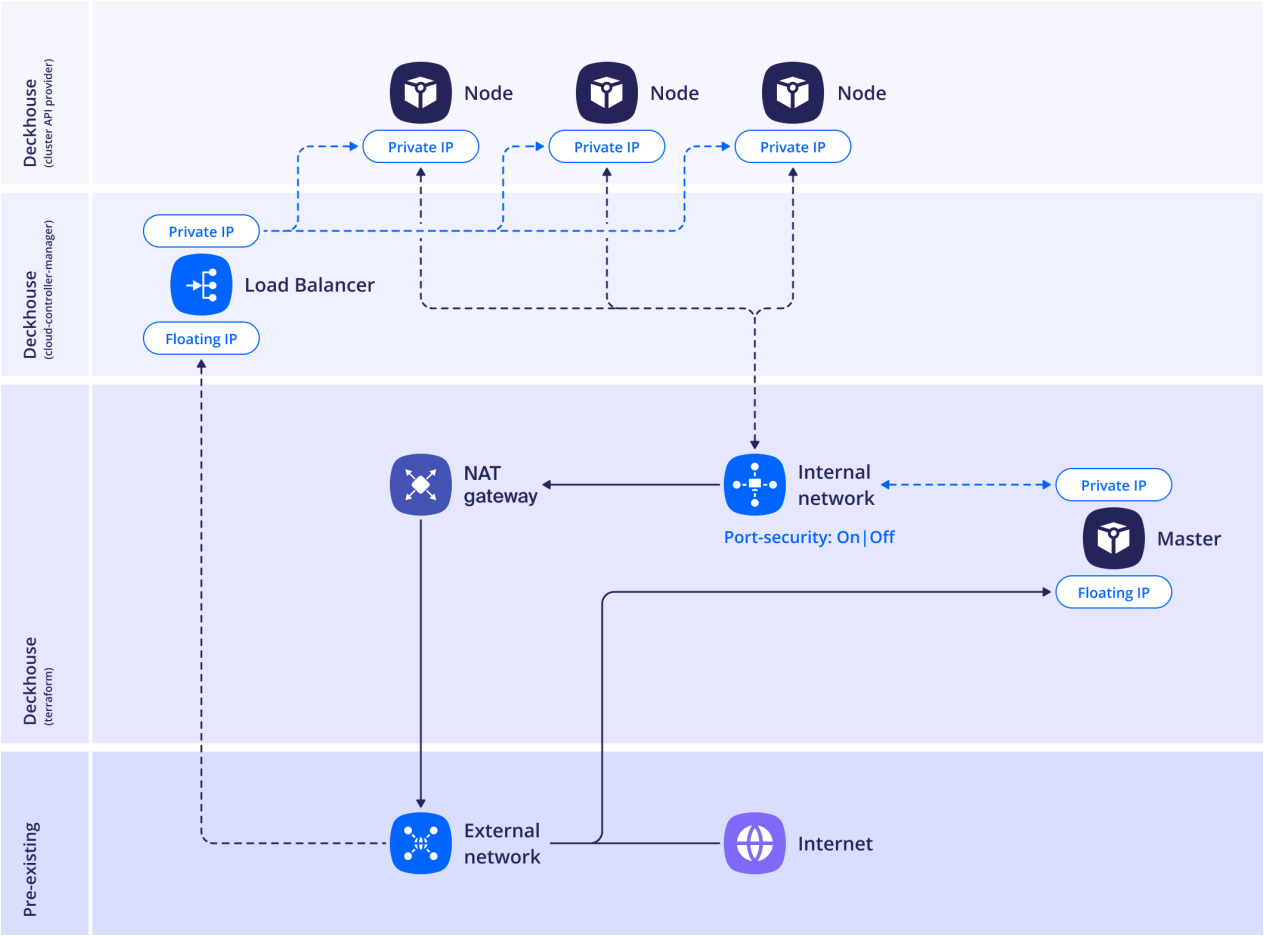
Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре OpenStack и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.

⚠

Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                 # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false        # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared               # Обязательный параметр.
bastion:
  zone: ru2-b                               # Необязательный параметр.
  volumeType: fast-ru-2b                   # Необязательный параметр.
  instanceClass:
    flavorName: m1.large                   # Обязательный параметр.
    imageName: ubuntu-20-04-cloud-amd64   # Обязательный параметр.
    rootDiskSize: 50                      # Обязательный параметр, по умолчанию 50 ГБ.
  additionalTags:
    severity: critical                     # Необязательный параметр.
```

```

    environment: production                                # Необязательный параметр.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                                  # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64                 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
    additionalTags:
      severity: critical
      environment: production
# Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
# (всегда используйте самый быстрый диск, предоставленный поставщиком).
volumeTypeMap:
  # Если указан rootDiskSize, этот тип тома также будет
  # использоваться для главного корневого тома.
  ru-1a: fast-ru-1a
  ru-1b: fast-ru-1b
  ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                                  # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64                 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                                  # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...

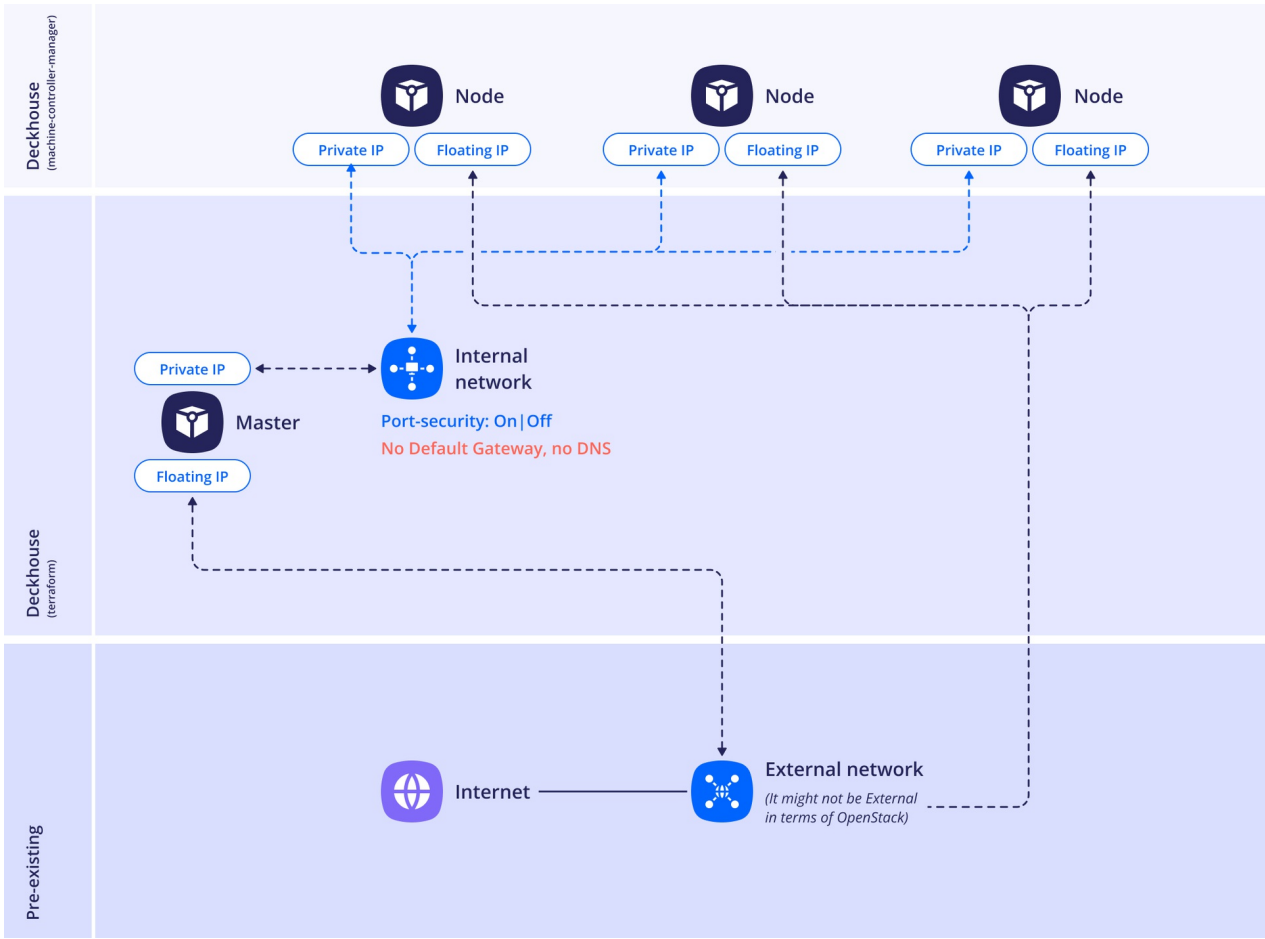
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

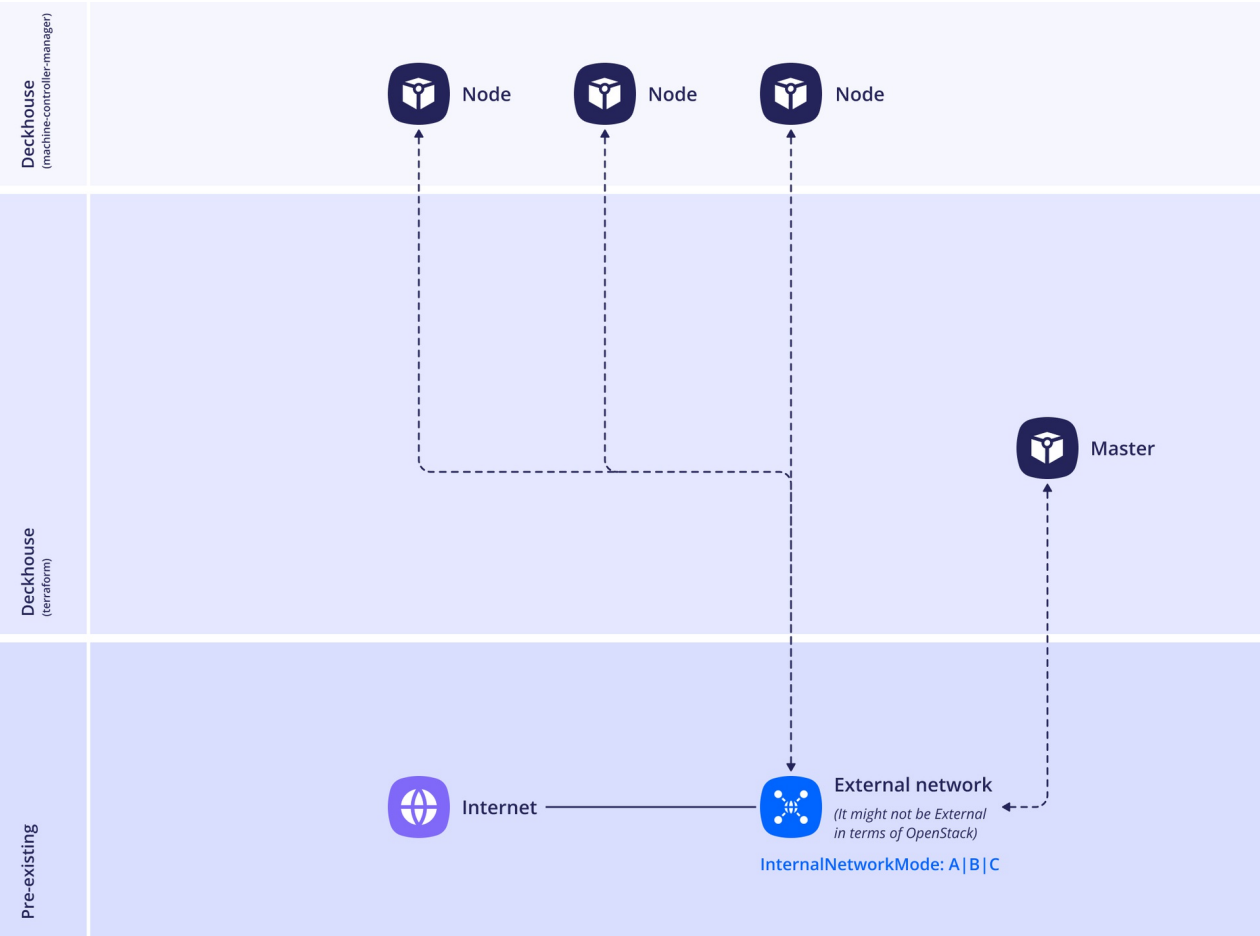
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                   # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                  # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

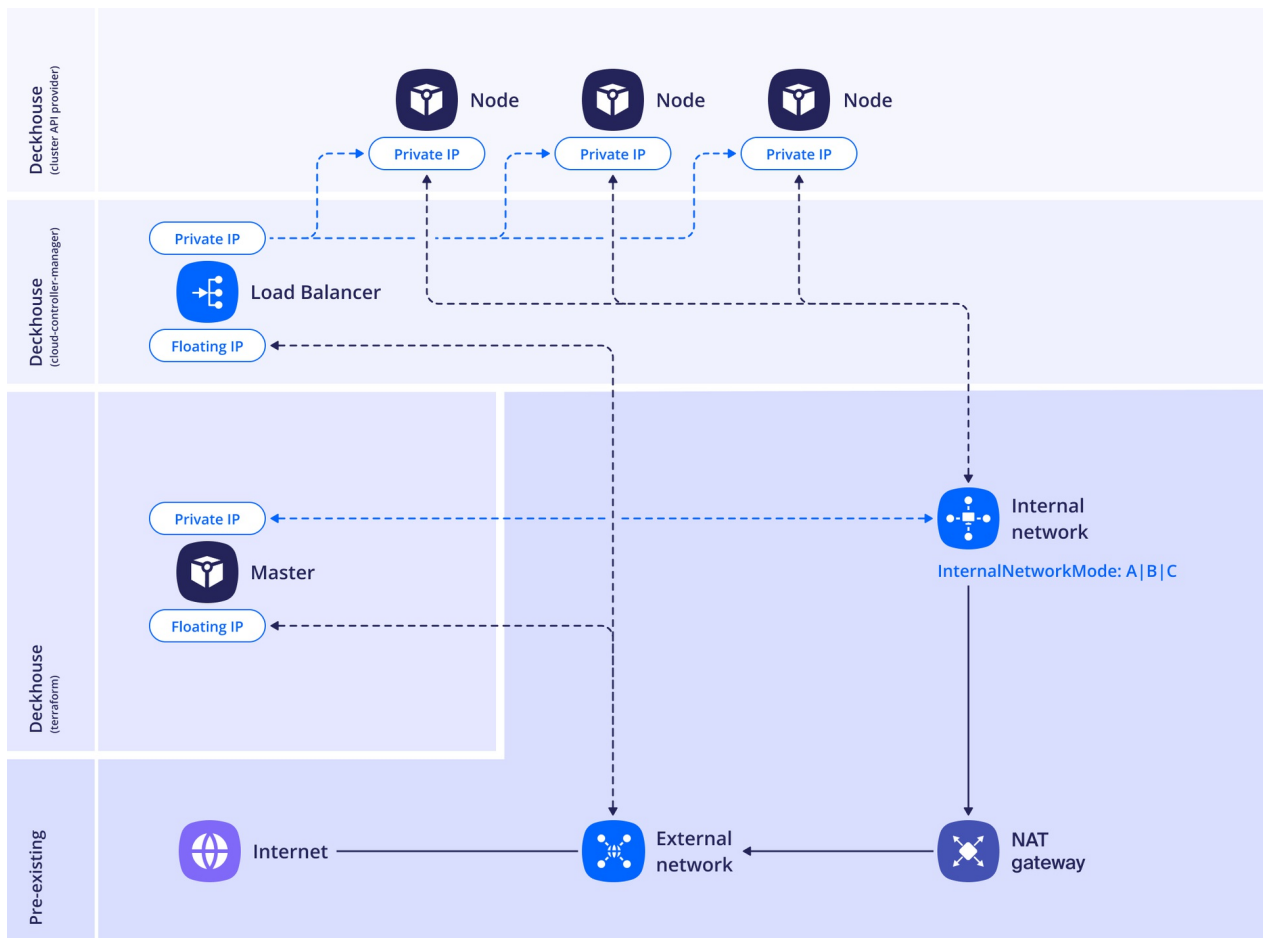
```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:


```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграции с OpenStack осуществляется с помощью ресурса OpenStackClusterConfiguration, который описывает конфигурацию облачного кластера в OpenStack и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера OpenStack — это `custom resource OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».



При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов OpenStack, необходимых для работы Deckhouse Kubernetes Platform в OpenStack:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим ВМ в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в OpenStack

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте `openrc`-файл, который содержит `credentials` для обращения к API OpenStack.

i Интерфейс получения `openrc`-файла может отличаться в зависимости от провайдера OpenStack. Если провайдер предоставляет стандартный интерфейс для OpenStack, скачать `openrc`-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него `openrc`-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из `openrc`-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

```
+-----+-----+-----+
| ID                                     | Name           | Is Public |
+-----+-----+-----+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | kol-high-iops  | True      |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops  | True      |
| 74101409-a462-4f03-872a-7de727a178b8 | kol-ssd        | True      |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd        | True      |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd            | True      |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1            | True      |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1            | True      |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph           | True      |
+-----+-----+-----+
```

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если OpenStack не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:

```

+-----+-----+
|
|
|-----+
| Field           | Value
|
+-----+-----+
|
|-----+
| checksum        | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+-----+-----+
|
|-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internalurl_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internalurl }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в OpenStack

Диск узла может быть локальным или сетевым. В терминологии OpenStack локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в OpenStack flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

i При создании узлов с типом CloudEphemeral в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -c disk
```

Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ OpenStack и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Офлайн-изменение размера диска

Если при изменении размера диска вы получаете следующую ошибку, необходимо уменьшить количество реплик StatefulSet до 0, подождать изменения размера дисков и вернуть обратно количество реплик, которое было до начала операции.

```
Warning VolumeResizeFailed 5s (x11 over 41s) external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Подключение и авторизация в Amazon Web Services

Требования



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Перед началом работы необходимо подготовить облачное окружение и обеспечить доступ к AWS-ресурсам для компонентов Deckhouse Kubernetes Platform (DKP), взаимодействующих с API AWS.

На всех виртуальных машинах, которые будут использоваться в составе кластера, должен быть установлен пакет `cloud-init`. После запуска необходимо убедиться, что активны следующие службы:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

Эти службы необходимы для корректной инициализации экземпляров EC2 и взаимодействия с инфраструктурными модулями DKP, такими как `cloud-provider-aws` и `machine-controller-manager`.

Доступ к AWS API

Для доступа к API AWS требуется IAM-пользователь с соответствующим набором прав. Необходимо создать JSON-спецификацию IAM-политики с разрешениями на работу с EC2, ELB, IAM, KMS и другими сервисами AWS.

Пример JSON-файла с конфигурацией необходимых прав:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "ec2:AllocateAddress",
        "ec2:AssociateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AttachVolume",
```

```

"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CreateInternetGateway",
"ec2:CreateKeyPair",
"ec2:CreateNATGateway",
"ec2:CreateRoute",
"ec2:CreateRouteTable",
"ec2:CreateSecurityGroup",
"ec2:CreateSubnet",
"ec2:CreateTags",
"ec2:CreateVolume",
"ec2:CreateVpc",
"ec2>DeleteInternetGateway",
"ec2>DeleteKeyPair",
"ec2>DeleteNATGateway",
"ec2>DeleteRoute",
"ec2>DeleteRouteTable",
"ec2>DeleteSecurityGroup",
"ec2>DeleteSubnet",
"ec2>DeleteTags",
"ec2>DeleteVolume",
"ec2>DeleteVpc",
"ec2:DescribeAccountAttributes",
"ec2:DescribeAddresses",
"ec2:DescribeAvailabilityZones",
"ec2:DescribeImages",
"ec2:DescribeInstanceAttribute",
"ec2:DescribeInstanceCreditSpecifications",
"ec2:DescribeInstances",
"ec2:DescribeInstanceTypes",
"ec2:DescribeInstanceTopology",
"ec2:DescribeInternetGateways",
"ec2:DescribeKeyPairs",
"ec2:DescribeNatGateways",
"ec2:DescribeNetworkInterfaces",
"ec2:DescribeRegions",
"ec2:DescribeRouteTables",
"ec2:DescribeSecurityGroups",
"ec2:DescribeSecurityGroupRules",
"ec2:DescribeSubnets",
"ec2:DescribeTags",
"ec2:DescribeVolumesModifications",
"ec2:DescribeVolumes",
"ec2:DescribeVpcAttribute",
"ec2:DescribeVpcClassicLink",
"ec2:DescribeVpcClassicLinkDnsSupport",
"ec2:DescribeVpcs",
"ec2:DetachInternetGateway",
"ec2:DetachVolume",
"ec2:DisassociateAddress",
"ec2:DisassociateRouteTable",
"ec2:ImportKeyPair",
"ec2:ModifyInstanceAttribute",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVolume",
"ec2:ModifyVpcAttribute",
"ec2:ReleaseAddress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress",
"ec2:RunInstances",
"ec2:TerminateInstances",
"ec2:DescribeVpcPeeringConnections",
"ec2:CreateVpcPeeringConnection",
"ec2>DeleteVpcPeeringConnection",
"ec2:AcceptVpcPeeringConnection",
"ec2:CreateNetworkInterface",
"ec2:DescribeNetworkInterfaceAttribute",
"ec2:ModifyNetworkInterfaceAttribute",
"ec2>DeleteNetworkInterface",
"ec2:DescribeNetworkInterfaces",
"elasticloadbalancing:AddTags",
"elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
"elasticloadbalancing:AttachLoadBalancerToSubnets",

```



```

        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing:CreateListener",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:CreateLoadBalancerListeners",
        "elasticloadbalancing:CreateLoadBalancerPolicy",
        "elasticloadbalancing:CreateTargetGroup",
        "elasticloadbalancing>DeleteListener",
        "elasticloadbalancing>DeleteLoadBalancer",
        "elasticloadbalancing>DeleteLoadBalancerListeners",
        "elasticloadbalancing>DeleteTargetGroup",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:DescribeListeners",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:DescribeLoadBalancerPolicies",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "elasticloadbalancing:DetachLoadBalancerFromSubnets",
        "elasticloadbalancing:ModifyListener",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:ModifyTargetGroup",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets",
        "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
        "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
        "iam:AddRoleToInstanceProfile",
        "iam:CreateInstanceProfile",
        "iam:CreateRole",
        "iam:CreateServiceLinkedRole",
        "iam>DeleteInstanceProfile",
        "iam>DeleteRole",
        "iam>DeleteRolePolicy",
        "iam:GetInstanceProfile",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "iam:ListInstanceProfilesForRole",
        "iam:ListRolePolicies",
        "iam:ListAttachedRolePolicies",
        "iam:PassRole",
        "iam:PutRolePolicy",
        "iam:RemoveRoleFromInstanceProfile",
        "iam:TagRole",
        "kms:DescribeKey",
        "sts:GetCallerIdentity"
    ],
    "Resource": "*"
}
]
}

```

Настройка IAM через веб-интерфейс AWS

1. Создайте политики:

- Перейдите в раздел «Identity and Access Management (IAM)» → «Policies».
- Нажмите «Create policy».
- Перейдите на вкладку «JSON» и вставьте подготовленную JSON-спецификацию.
- Нажмите «Next: Tags», затем «Next: Review».
- Укажите имя политики, например D8CloudProviderAWS.
- Нажмите «Create policy».

2. Создайте пользователя:

- Перейдите в «IAM» → «Users».
- Нажмите «Add users».
- В поле «User name» укажите, например, deckhouse.
- В разделе «Select AWS credential type» выберите опцию «Access key» → «Programmatic access».
- Нажмите «Next: Permissions».

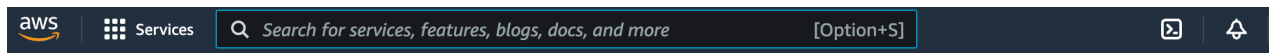
3. Привяжите политики к пользователю:

- Выберите «Attach existing policies directly».
- Введите имя ранее созданной политики в поле фильтрации и установите флажок рядом с нужной политикой.
- Нажмите «Next: Tags», затем «Next: Review».
- Нажмите «Create user».



Сохраните полученные `Access key ID` и `Secret access key`. Эти данные потребуются для настройки DKP.

Проверьте, есть ли у вашей учетной записи (и, соответственно, у созданного пользователя) доступ к нужным регионам. Для этого выберите необходимый регион в выпадающем списке в правом верхнем углу. Если произойдет переключение в выбранный регион, доступ к региону есть. Если доступа к региону нет, вы получите следующее сообщение (может отличаться):



Enable Middle East (UAE) region

You can control whether users in your AWS account can access resources in this region. When you enable access, AWS performs actions to prepare your account in that region, such as distributing your IAM resources to the region. You can use this AWS Region after AWS finishes preparing it. For most accounts, preparation is completed in a few minutes. [Learn more](#)

Enabling a region is free, but your users might create or use resources that result in AWS fees. [Learn more](#)

Continue to the account page and enable this region if you are an administrator for this account or [user with permissions to enable access to the region](#). If you are unsure whether you have the correct permissions, contact your administrator to enable this region on your behalf.

Cancel

Continue

В этом случае нажмите «Continue», чтобы разрешить использование региона.

Настройка IAM через AWS CLI

Также IAM можно настроить через интерфейс командной строки. Для этого выполните следующие шаги:

1. Сохраните JSON-спецификацию в файл `policy.json`:

```
cat > policy.json << EOF
<Policy JSON spec>
EOF
```

2. Создайте новую Policy с именем `D8CloudProviderAWS` и примечанием `ARN`, используя JSON-спецификацию из файла `policy.json`:

```
aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
```

Пример вывода:

```
{
  "Policy": {
    "PolicyName": "D8CloudProviderAWS",
    "PolicyId": "AAA",
    "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2020-08-27T02:52:06+00:00",
    "UpdateDate": "2020-08-27T02:52:06+00:00"
  }
}
```

3. Создайте нового пользователя:

```
aws iam create-user --user-name deckhouse
```

Пример вывода:

```
{
  "User": {
    "Path": "/",
    "UserName": "deckhouse",
    "UserId": "AAAXXX",
    "Arn": "arn:aws:iam::123:user/deckhouse",
    "CreateDate": "2020-08-27T03:05:42+00:00"
  }
}
```

4. Разрешите доступ к API и сохраните `AccessKeyId` и `SecretAccessKey`:

```
aws iam create-access-key --user-name deckhouse
```

Пример вывода:

```
{
  "AccessKey": {
    "UserName": "deckhouse",
    "AccessKeyId": "XXXXYY",
    "Status": "Active",
    "SecretAccessKey": "ZZZzzz",
    "CreateDate": "2020-08-27T03:06:22+00:00"
  }
}
```

5. Привяжите политики к пользователю. Объедините `User` и `Policy`:

```
aws iam attach-user-policy --user-name username --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
```

Настройка IAM через Terraform

Пример конфигурации ресурсов для Terraform:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name      = "D8CloudProviderAWS"
  path      = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON-спецификация Policy>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user          = aws_iam_user.user.name
  policy_arn    = aws_iam_policy.policy.arn
}
```

Использование своей IAM-роли для узлов (iamNodeRole)

Параметр `iamNodeRole` в ресурсе `AWSClusterConfiguration` позволяет переопределить IAM-роль по умолчанию, которую DKP назначает всем EC2-инстансам узлов кластера.

По умолчанию DKP создаёт и назначает IAM-роль с именем `<prefix>-node`, где `<prefix>` — это значение глобального параметра `cloud.prefix`. Эта роль включает минимально необходимый набор прав для работы узлов.

Если же вам необходимо:

- использовать предварительно созданную IAM-роль, управляемую вручную;
- предоставить узлам расширенные полномочия, например, доступ к специфичным сервисам AWS;

— вы можете указать имя IAM-роли в параметре `iamNodeRole`.

i Указанная роль должна включать все права, требуемые DKP, иначе узлы не смогут полноценно функционировать. В частности, она обязательно должна включать права роли `<prefix>-node`, даже если они назначаются вручную.

Подробнее о IAM-ролях для AWS EC2 можно прочитать в документации AWS.

Дополнительные ресурсы

- Документация модуля `cloud-provider-aws`

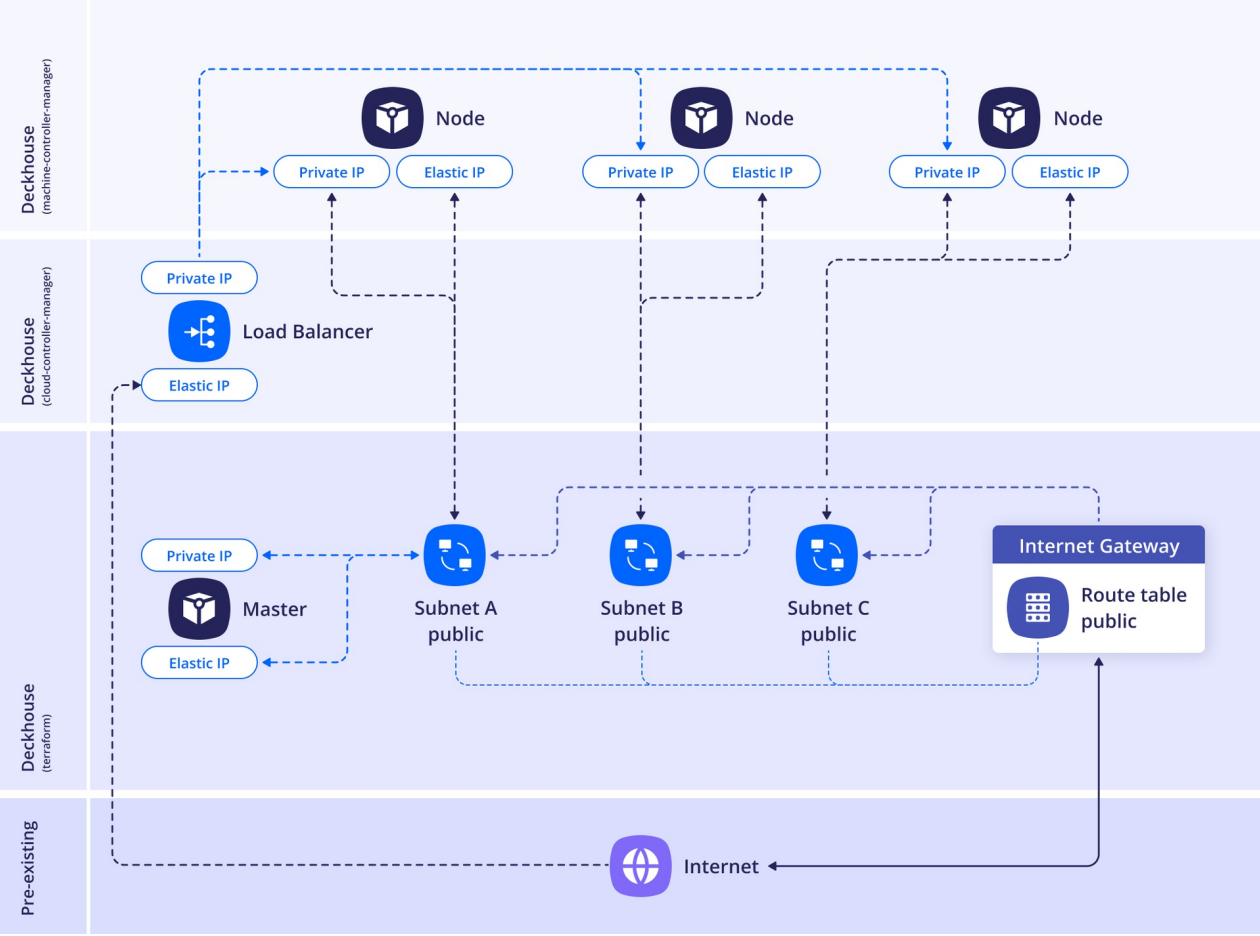
Схемы размещения и настройка Amazon Web Services

Данный раздел описывает схемы размещения кластера в инфраструктуре AWS и связанные с ними параметры. Выбор схемы (layout) влияет на способ назначения публичных IP-адресов, работу NAT и возможность подключения к узлам.

WithoutNAT

Рекомендуемая схема размещения.

Каждому узлу назначается публичный IP-адрес (Elastic IP), NAT-шлюз не используется. Такая схема обеспечивает прямой доступ к узлам по публичным IP-адресам и позволяет упростить маршрутизацию исходящего трафика.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  # Количество master-узлов.
  # Если указано больше одного master-узла, то etcd-кластер соберется автоматически.
  replicas: 1
  instanceClass:
    # Тип используемого инстанса.
    instanceType: m5.xlarge
    # ID образа виртуальной машины в Amazon.
    # Каталог AMI в консоли AWS: EC2 -> AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Размер диска для виртуальной машины master-узла.
    diskSizeGb: 30
    # Используемый тип диска для виртуальной машины master-узла.
    diskType: gp3
nodeGroups:
- name: mydb
  nodeTemplate:
    labels:
      node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: torpedo

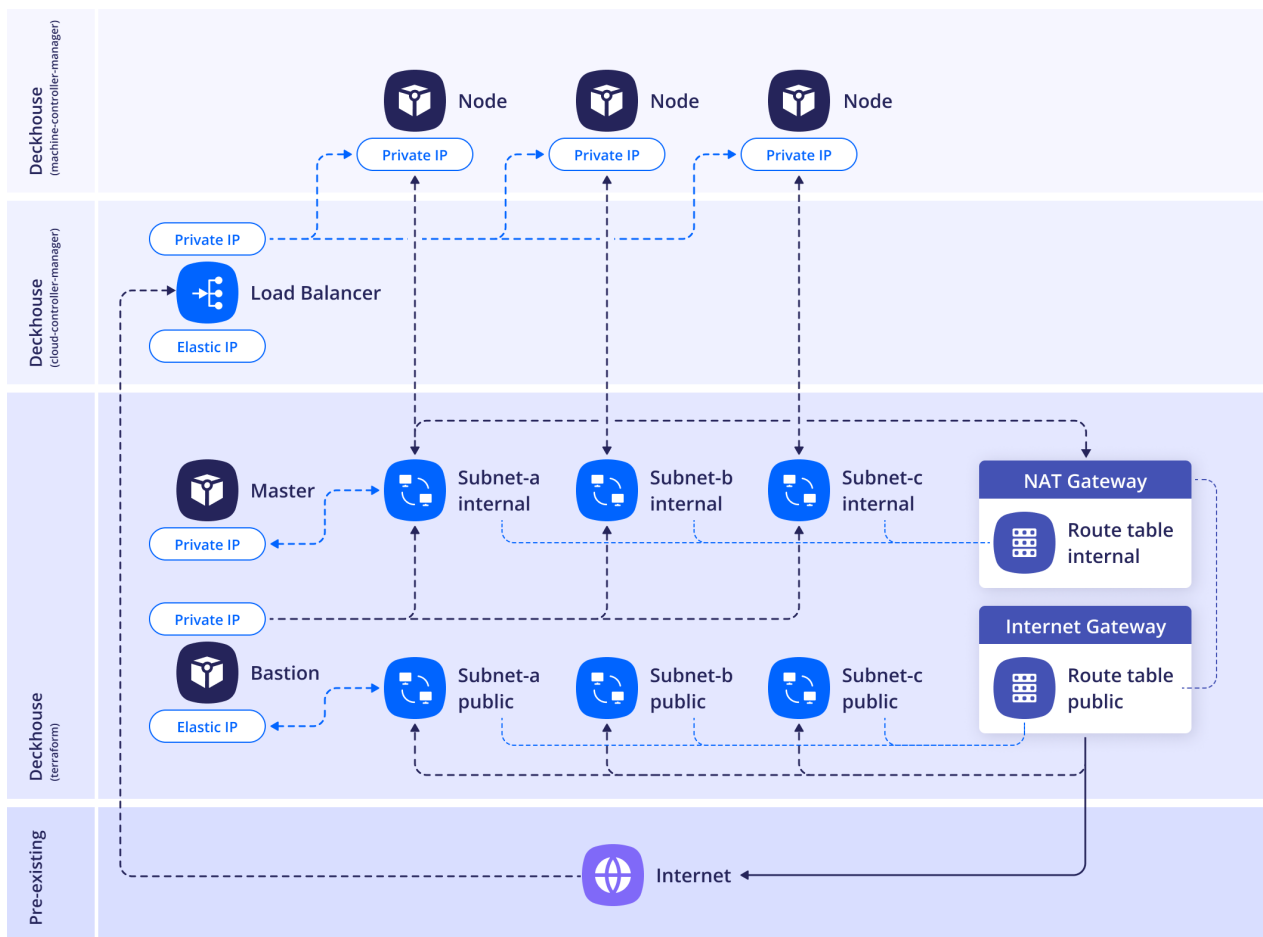
```

WithNAT



В данной схеме NAT Gateway всегда создается в зоне `a`. Если узлы кластера размещены в других зонах, то при сбое зоны `a` кластер может стать недоступным. Bastion-хост обязателен для подключения к узлам.

В этой схеме размещения NAT Gateway используется для выхода в интернет, а публичные IP-адреса узлам не присваиваются. Доступ к узлам возможен только через bastion-хост, размещаемый в отдельной подсети.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Количество master-узлов.
  # Если указано больше одного master-узла, etcd-кластер соберется автоматически.
  replicas: 1
  instanceClass:
    # Тип используемого инстанса.
    instanceType: m5.xlarge
    # ID образа виртуальной машины в Amazon.
    # Каталог AMI в консоли AWS: EC2 -> AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Размер диска для виртуальной машины master-узла.
    diskSizeGb: 30
    # Используемый тип диска для виртуальной машины master-узла.
    diskType: gp3
nodeGroups:
- name: mydb
  nodeTemplate:
    labels:
      node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  team: torpedo

```

Назначение AWSClusterConfiguration

Ресурс AWSClusterConfiguration описывает параметры кластера и используется Deckhouse Kubernetes Platform (DKP) для:

- задания схемы размещения и сетевых CIDR;
- конфигурации master- и рабочих узлов;
- указания параметров подключения к AWS API (ключи доступа, регион);
- назначения общих и специфичных тегов;
- описания настроек bastion-хоста (в схеме WithNAT).

Обязательные поля:

- `apiVersion` — должен быть `deckhouse.io/v1`;
- `kind` — всегда `AWSClusterConfiguration`.

Пример заголовка ресурса:

```

apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration

```

Чтобы отредактировать этот ресурс в работающем кластере, выполните команду:

```
d8 system edit provider-cluster-configuration
```

После внесения изменений их необходимо применить с помощью команды:

```
dhctl converge
```

Внутренняя адресация и подсети

Параметр `nodeNetworkCIDR` определяет диапазон адресов, который будет распределен по зонам доступности. Этот диапазон должен соответствовать или быть вложенным в `vpcNetworkCIDR`. Подсети автоматически создаются на основании количества зон региона.

Пример:

```
nodeNetworkCIDR: 10.241.1.0/20
vpcNetworkCIDR: 10.241.0.0/16
```

Группы безопасности

Группы безопасности (security groups) в AWS используются для управления входящим и исходящим сетевым трафиком на виртуальные машины. В DKP они позволяют:

- разрешить подключение к узлам кластера с других подсетей;
- открыть доступ к приложениям, размещённым на статических узлах;
- ограничить или разрешить доступ к внешним ресурсам в соответствии с требованиями безопасности.



DKP не создаёт группы безопасности автоматически. В конфигурации кластера следует указывать уже существующие security groups, созданные вручную через AWS Console или иным способом.

Дополнительные группы безопасности можно назначить в следующих случаях:

Тип узлов	Где указывать
Master-узлы	В поле <code>masterNodeGroup.instanceClass.additionalSecurityGroups</code> ресурса <code>AWSClusterConfiguration</code>
Статические worker-узлы	В поле <code>nodeGroups[].instanceClass.additionalSecurityGroups</code> того же ресурса
Эфемерные узлы	В объекте <code>AWSInstanceClass</code> , в поле <code>spec.additionalSecurityGroups</code>

Во всех случаях параметр `additionalSecurityGroups` принимает массив строк — имен (ID) групп безопасности в AWS.

Если установлено значение `disableDefaultSecurityGroup: true`, то группы по умолчанию создаваться не будут.

При использовании `disableDefaultSecurityGroup: true` необходимо самостоятельно создать все необходимые группы безопасности для доступа к узлам кластера. Кроме того, необходимо явно указать их в следующих параметрах:

- `additionalSecurityGroups` в секции `masterNodeGroup` ресурса `AWSClusterConfiguration`;
- `additionalSecurityGroups` в ресурсе `AWSInstanceClass`;
- `additionalSecurityGroups` в секции `nodeGroups.instanceClass`.

Для настройки групп, используемых балансировщиками нагрузки, укажите их через аннотацию `service.beta.kubernetes.io/aws-load-balancer-security-groups`.

Настройка пирингового соединения между VPC

Для примера рассмотрим настройку пирингового соединения между двумя условными VPC — `vpc-a` и `vpc-b`.



IPv4 CIDR у обоих VPC должен различаться.

Для настройки выполните следующие шаги:

1. Перейдите в регион, где работает `vpc-a`.
2. Нажмите «VPC» → «VPC Peering Connections» → «Create Peering Connection» и настройте пиринговое соединение:
 - Name: `vpc-a-vpc-b`.
 - Заполните «Local» и «Another VPC».
3. Перейдите в регион, где работает `vpc-b`.
4. Нажмите «VPC» → «VPC Peering Connections».
5. Выделите созданное соединение и выберите «Action Accept Request».
6. Для `vpc-a` добавьте во все таблицы маршрутизации маршруты до CIDR `vpc-b` через пиринговое соединение.
7. Для `vpc-b` добавьте во все таблицы маршрутизации маршруты до CIDR `vpc-a` через пиринговое соединение.

Настройка доступа через bastion-хост

Для подключения к узлам в частных подсетях используйте параметр `withNAT.bastionInstance` в `AWSClusterConfiguration`. Bastion-хост заказывается вместе с инфраструктурой по заданным параметрам `instanceClass`.

Поддерживаются сценарии:

- bastion-хост уже создан во внешней VPC:
 1. Создайте базовую инфраструктуру кластера — `dhctl bootstrap-phase base-infra`.
 2. Настройте пиринговое соединение между внешней и свежесозданной VPC.
 3. Продолжите установку с указанием bastion-хоста — `dhctl bootstrap --ssh-bastion...`.
- bastion-хост требуется поставить в свежесозданной VPC:
 1. Создайте базовую инфраструктуру кластера — `dhctl bootstrap-phase base-infra`.
 2. Запустите вручную bastion-хост в подсети `<prefix>-public-0`.
 3. Продолжите установку с указанием bastion-хоста — `dhctl bootstrap --ssh-bastion...`.

Создание кластера в новом VPC с доступом через имеющийся bastion-хост

1. Выполните bootstrap базовой инфраструктуры кластера:

```
dhctl bootstrap-phase base-infra --config config
```

2. Настройте пиринговое соединение по инструкции выше.
3. Продолжите установку кластера. На вопрос про кеш Terraform ответьте `y`:

```
dhctl bootstrap --config config --ssh-...
```

Создание кластера в новом VPC и развертывание bastion-хоста для доступа к узлам

1. Выполните bootstrap базовой инфраструктуры кластера:

```
dhctl bootstrap-phase base-infra --config config
```

2. Запустите вручную bastion-хост в подсети `<prefix>-public-0`.
3. Продолжите установку кластера. На вопрос про кеш Terraform ответьте `y`:

```
dhctl bootstrap --config config --ssh-...
```

Использование существующего VPC (existingVPCID)

Параметр `existingVPCID` в ресурсе `AWSClusterConfiguration` позволяет использовать уже существующий VPC для развертывания кластера DKP, вместо создания нового VPC автоматически.

Этот параметр может быть полезен в случаях, когда:

- инфраструктура AWS уже частично развернута;
- необходимо интегрироваться с другими сервисами или ресурсами, размещёнными в этом VPC;
- политика безопасности или архитектурные требования запрещают автоматическое создание VPC.



Если в существующем VPC уже есть Internet Gateway, попытка развертывания базовой инфраструктуры завершится ошибкой. В текущей версии DKP не поддерживается повторное использование уже существующего Internet Gateway.

Совместимость с другими параметрами:

- Если указан `existingVPCID`, не указывайте `vpcNetworkCIDR` — это взаимоисключающие параметры.
- Параметр `nodeNetworkCIDR` можно (и желательно) указывать — он будет вложен в существующий VPC.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-aws`
- Кастомные ресурсы модуля `cloud-provider-aws`

Хранилище и балансировка нагрузки в Amazon Web Services

Этот раздел охватывает дополнительные аспекты интеграции Deckhouse Kubernetes Platform (DKP) с AWS:

- Подключение облачных дисков через CSI;
- Автоматическое создание `StorageClass`;
- Использование `LoadBalancer`;
- Доступ через bastion-хост;
- Подключение вручную созданных `CloudStatic`-узлов.

Хранилище (CSI и StorageClass)

DKP обеспечивает интеграцию с хранилищем AWS через CSI. Это позволяет кластеру автоматически заказывать и подключать диски к своим узлам.

`StorageClass` создаётся автоматически для следующих типов дисков:

- `gp3`;
- `gp2`;
- `sc1`;

- `st1`.

Также поддерживаются диски `io1` и `io2` с настройками IOPS и throughput в `ModuleConfig`.

Для исключения из кластера ненужных StorageClass укажите фильтры в `settings.storageClass.exclude`:

```
settings:
  storageClass:
    exclude:
      - sc.*
      - st1
```

Можно явно указать класс и настроить для StorageClass параметры, включая `iops`, `throughput`, `type`:

```
settings:
  storageClass:
    provision:
      - name: fast-io
        type: io2
        iopsPerGB: "50"
```

Увеличение объема тома (volume)

Чтобы изменить размер тома (например, при нехватке дискового пространства), выполните следующие шаги:

1. Измените параметр `spec.resources.requests.storage` в соответствующем объекте `PersistentVolumeClaim`. Операция выполняется автоматически и обычно занимает не более одной минуты. За ходом процесса можно наблюдать с помощью команды:

```
d8 k describe pvc <имя-claim>
```

2. После изменения объема подождите не менее 6 часов и убедитесь, что статус volume — `in-use` или `available`. Только после этого возможны повторные изменения. Подробнее в официальной документации AWS.

Балансировка нагрузки

DKP поддерживает работу с LoadBalancer Service через AWS Load Balancer Controller.

Для контроля создания ресурсов в AWS используются аннотации объекта `Service`:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-12345,subnet-67890"
```

- `service.beta.kubernetes.io/aws-load-balancer-type` — при значении `none` создаётся только целевая группа, без самого LoadBalancer.
- `service.beta.kubernetes.io/aws-load-balancer-backend-protocol` — применяется только в связке с `aws-load-balancer-type: none` и определяет протокол взаимодействия с целевой группой. Поддерживаются следующие значения:
 - `tcp` (значение по умолчанию);
 - `tls`;
 - `http`;
 - `https`.

i При изменении значения этой аннотации `cloud-controller-manager` попытается пересоздать целевую группу. Если она уже используется в связке с NLB или ALB, удалить её не удастся, и контроллер будет бесконечно повторять попытку. Чтобы избежать этого, необходимо вручную отсоединить балансировщик от группы.

Если Ingress-узлы есть не во всех зонах, укажите явно подсети в `aws-load-balancer-subnets`.

Настройка балансировщика при отсутствии Ingress-узлов в некоторых зонах

Если Ingress-узлы присутствуют не во всех зонах AWS, необходимо явно указать, какие подсети использовать для балансировщика. Это делается с помощью аннотации `service.beta.kubernetes.io/aws-load-balancer-subnets`:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-foo,subnet-bar"
```

Это особенно важно при ручной настройке Ingress-контроллеров или нестандартной схеме размещения узлов.

Чтобы получить список текущих подсетей, используемых в установке DKP, выполните команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse \
  -- deckhouse-controller module values cloud-provider-aws -o json | \
  jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

Подключение CloudStatic-узлов

Для подключения созданных вручную EC2-инстансов к кластеру DKP:

1. Добавьте IAM-роль `<prefix>-node`.
2. Добавьте группу безопасности `<prefix>-node`.
3. Укажите теги:

```
"kubernetes.io/cluster/<cluster_uuid>" = "shared"
"kubernetes.io/cluster/<prefix>" = "shared"
```

Узнать `cluster_uuid` можно с помощью команды:

```
d8 k -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
```

Узнать `prefix` можно с помощью команды:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | \
  jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix
```

Дополнительные ресурсы

- Параметры модуля `cloud-provider-aws`

Интеграция со службами Amazon Web Services

Работа с Amazon Elastic Container Registry (ECR)

Чтобы узлы кластера имели доступ к частным репозиториям Amazon ECR:

1. Определите права на чтение образов в политиках репозитория. Важно, чтобы в `Principal` был указан существующий IAM-Role (IAM-роль), привязанная к узлам Deckhouse Kubernetes Platform (DKP).

Пример политики:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RepositoryRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::xxx:role/xxx-node"
      },
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:DescribeRepositories",
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:ListImages",
        "ecr:ListTagsForResource"
      ]
    }
  ]
}
```

2. Настройте эту политику в AWS Console: Amazon ECR → Private registry → Repositories → требуемый репозиторий → Permissions.
3. Добавьте `ecr:GetAuthorizationToken` в `additionalRolePolicies` в `AWSClusterConfiguration`, чтобы узлы могли автоматически получать токен доступа к образам:

```
additionalRolePolicies:
- ecr:GetAuthorizationToken
- ecr:BatchGetImage
- ecr:DescribeRepositories
```

Параметр `additionalRolePolicies` позволяет расширить набор IAM-действий, назначаемых EC2-инстансам, управляемым DKP. Это особенно полезно, если требуется доступ к:

- Amazon ECR;
- Amazon S3;
- другим сервисам AWS, требующим специфических прав.

Если параметр не задан, IAM-роли будут содержать только базовые действия:

```
ec2:DescribeTags
ec2:DescribeInstances
```

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-aws`

Подключение и авторизация в Microsoft Azure

Требования



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для корректной работы Deckhouse Kubernetes Platform (DKP) с Microsoft Azure необходимо:

- Убедиться, что используемый регион поддерживает Availability Zones.
- На всех виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска ВМ должны быть активны службы:
 - `cloud-config.service`;
 - `cloud-final.service`;
 - `cloud-init.service`.

Доступ к Azure API

Для управления ресурсами Azure из DKP требуется сервисный аккаунт с ролью `Contributor` в рамках нужной подписки. Выполните следующие шаги:

1. Установите Azure CLI и авторизуйтесь:

```
export SUBSCRIPTION_ID=$(az login | jq -r '[0].id')
```

2. Создайте сервисный аккаунт:

```
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --name "DeckhouseCANDI"
```

Пример вывода команды:

```
{
  "appId": "<CLIENT_ID>",
  "displayName": "DeckhouseCANDI",
  "password": "<CLIENT_SECRET>",
  "tenant": "<TENANT_ID>"
}
```

Эти параметры необходимо указать в объекте `AzureClusterConfiguration`:

Поле	Значение из вывода команды
<code>clientId</code>	<code>appId</code>
<code>clientSecret</code>	<code>password</code>
<code>tenantId</code>	<code>tenant</code>

3. Авторизуйтесь в Azure CLI под созданным сервисным аккаунтом:

```
az login --service-principal -u <CLIENT_ID> -p <CLIENT_SECRET> --tenant <TENANT_ID>
```



Срок действия `clientSecret` по умолчанию — 1 год. Автоматическое продление не поддерживается. Чтобы задать больший срок действия, воспользуйтесь официальной документацией Azure.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-azure`

Схемы размещения и настройка Microsoft Azure

Данный раздел описывает схему размещения кластера в инфраструктуре Azure и связанные с ней параметры.

Standard

Standard — поддерживаемая схема размещения:

- Для кластера создаётся отдельная ресурсная группа (resource group).
- Каждому инстансу по умолчанию выделяется внешний IP-адрес (используется только для выхода в интернет).
- Для SNAT используется до 64 000 портов на один IP-адрес.
- Поддерживается NAT Gateway с возможностью тарификации — позволяет использовать статические публичные IP-адреса для SNAT.
- Публичные IP могут быть назначены как на master-узлы, так и на узлы, созданные через Terraform.
- При отсутствии внешнего IP у master-узла требуется bastion-хост и VNet-пиринг между кластерами.
- Поддерживается пиринг между VNet кластера и другими VNet.

Пример конфигурации размещения:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"      # Обязательный параметр.
vnetCIDR: 10.50.0.0/16                 # Обязательный параметр.
subnetCIDR: 10.50.0.0/24               # Обязательный параметр.
standard:
  natGatewayPublicIpCount: 1           # Необязательный параметр (по умолчанию 0).
masterNodeGroup:
  replicas: 1
  zones: ["1"]                        # Необязательный параметр (по умолчанию ["1"]).
instanceClass:
  machineSize: Standard_F4            # Обязательный параметр.
  diskSizeGb: 32
  urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140 # Обязательный параметр.
  enableExternalIP: false             # Необязательный параметр (по умолчанию true).
provider:
  subscriptionId: "<SUBSCRIPTION_ID>" # Обязательный параметр.
  clientId: "<CLIENT_ID>"            # Обязательный параметр.
  clientSecret: "<CLIENT_SECRET>"    # Обязательный параметр.
  tenantId: "<TENANT_ID>"            # Обязательный параметр.
  location: "westeurope"              # Обязательный параметр.
peeredVNETs:                           # Необязательный параметр.
- resourceName: kube-bastion          # Обязательный параметр.
  vnetName: kube-bastion-vnet         # Обязательный параметр.
```

Хранилище и балансировка нагрузки в Microsoft Azure

Хранилище

При работе с Azure Deckhouse Kubernetes Platform (DKP) автоматически создаёт следующие StorageClass:

Имя	Тип диска
managed-standard	Standard_LRS
managed-standard-ssd	StandardSSD_LRS
managed-premium	Premium_LRS

Дополнительно можно:

- Отключить ненужные классы хранилища через параметр `exclude`.
- Создать свои StorageClass с нужными параметрами пропускной способности и IOPS.

Пример настройки в ModuleConfig:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
        - name: managed-ultra-ssd
          type: UltraSSD_LRS
          diskIOPSReadWrite: 600
          diskMBpsReadWrite: 150
      exclude:
        - managed-standard.*
        - managed-premium

```

Балансировка нагрузки

DKP автоматически создает ресурсы LoadBalancer в Azure при использовании Kubernetes-сервисов типа LoadBalancer.

Дополнительные особенности:

- Поддерживается использование NAT Gateway — можно явно задать количество публичных IP-адресов для SNAT через параметр `natGatewayPublicIpCount`.
- Возможна настройка пиринга VNet, в том числе с bastion-хостом или другими VNet в облаке.
- Поддерживаются Service Endpoints — безопасное и прямое подключение к сервисам Azure без использования публичных IP. Подробнее – в разделе Интеграция с сервисами Azure.

Дополнительные ресурсы

- Параметры модуля `cloud-provider-azure`
- Настройки провайдера модуля `cloud-provider-azure`

Интеграция со службами Microsoft Azure

Deckhouse Kubernetes Platform (DKP) использует возможности облака Azure для полноценной интеграции с Kubernetes. При работе с кластером в Azure автоматически:

- Создаются сетевые маршруты для сети PodNetwork;
- Настраиваются внешние балансировщики нагрузки (LoadBalancer) для сервисов Kubernetes;
- Удаляются из кластера узлы, отсутствующие в облаке;
- Обновляются метаданные узлов в соответствии с текущей конфигурацией;
- Заказываются диски для узлов через CSI;
- Подключается необходимая CNI-сеть (используется simple bridge);
- Становятся доступны описания виртуальных машин в формате AzureInstanceClass для последующего использования в NodeGroup.

i

Весь исходящий трафик из кластера проходит через балансировщики нагрузки. Если ни один из них не настроен для работы с UDP, весь исходящий UDP-трафик будет блокироваться, что может повлиять на работу NTP (`ntpd`, `chrony` и др.). Решение: вручную добавить правило на любой UDP-порт в существующий LoadBalancer или создать Kubernetes-сервис типа LoadBalancer с UDP-портом.

Поддержка Service Endpoints

Поддерживаются подключения к сервисам Azure через Service Endpoints, которые:

- Позволяют осуществлять подключение к сервисам Azure без использования публичных IP-адресов.
- Работают по оптимизированному маршруту через магистральную сеть Azure.
- Упрощают контроль доступа и повышают безопасность.

Список поддерживаемых Service Endpoints:

```

Microsoft.AzureActiveDirectory
Microsoft.AzureCosmosDB
Microsoft.ContainerRegistry
Microsoft.CognitiveServices
Microsoft.EventHub
Microsoft.KeyVault
Microsoft.ServiceBus
Microsoft.Sql
Microsoft.Storage
Microsoft.Storage.Global
Microsoft.Web

```

Укажите необходимые сервисы в параметре `serviceEndpoints` объекта `AzureClusterConfiguration`.

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-azure`
- Настройки провайдера модуля `cloud-provider-azure`
- Кастомные ресурсы модуля `node-manager`

Подключение и авторизация в Google Cloud Platform

Для управления ресурсами в Google Cloud с помощью Deckhouse Kubernetes Platform необходимо создать Service Account.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Создание Service Account

Подробную инструкцию по созданию Service Account можно найти в документации провайдера.



Созданный `service account key` невозможно восстановить, только удалить и создать новый.

Настройка через Google Cloud Console

Перейдите в Google Cloud Console, выберите проект и создайте новый Service Account (также можно выбрать уже существующий).

Созданному Service Account должны быть присвоены несколько обязательных ролей:

```
Compute Admin
Service Account User
Network Management Admin
```

Роли можно присвоить на этапе создания Service Account либо изменить.

Чтобы получить `service account key` в JSON-формате, на странице в колонке «Actions» нажмите на три вертикальные точки и выберите «Manage keys». Затем нажмите «Add key» → «Create new key» → «Key type» → «JSON».

Настройка через Google Cloud CLI

Установите и инициализируйте Google Cloud CLI, следуя официальной инструкции.

Для создания Service Account через интерфейс командной строки выполните следующие шаги:

1. Экспортируйте переменные окружения:

```
export PROJECT_ID=sandbox
export SERVICE_ACCOUNT_NAME=deckhouse
```

2. Выберите project:

```
gcloud config set project $PROJECT_ID
```

3. Создайте Service Account:

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

4. Присвойте роли созданному Service Account:

```
for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
--role=${role}; done
```

Список необходимых ролей:

```
roles/compute.admin
roles/iam.serviceAccountUser
roles/networkmanagement.admin
```

5. Выполните проверку ролей Service Account:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
--filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Создайте service account key:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

Использование созданного Service Account

Полученный service account key указывается в секции `provider.serviceAccountJSON`: "`<SERVICE_ACCOUNT_JSON>`" ресурса `GCPClusterConfiguration`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
instanceClass:
  machineType: n1-standard-4
  image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
  diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
instanceClass:
  machineType: n1-standard-4
  image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
  diskSizeGb: 50
  additionalNetworkTags:
  - tag1
  additionalLabels:
    kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-gcp`

Схемы размещения и настройка Google Cloud Platform

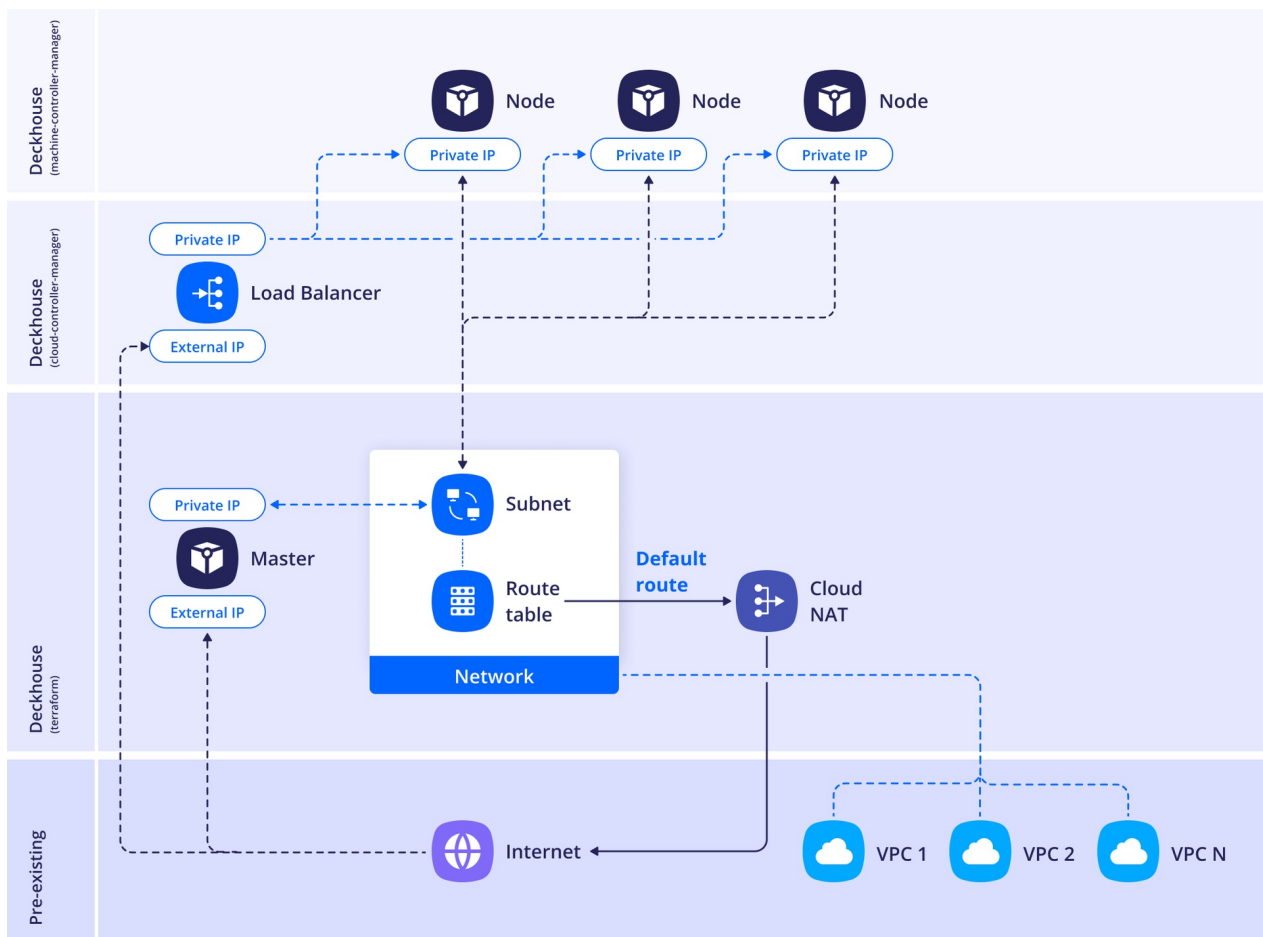
Схемы размещения

Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре Google Cloud Platform (GCP) и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

DKP поддерживает две схемы размещения ресурсов в облаке GCP.

Standard

- Для кластера создается отдельная VPC с Cloud NAT.
- Узлы в кластере не имеют публичных IP-адресов.
- Публичные IP-адреса можно назначить на статические и master-узлы:
 - при этом будет использоваться One-to-One NAT для отображения публичного IP-адреса в IP-адрес узла (следует помнить, что CloudNAT в этом случае использоваться не будет).
- Если master-узел не имеет публичного IP-адреса, для установки и доступа в кластер необходим дополнительный инстанс с публичным IP-адресом (например, bastion-хост).
- Между VPC кластера и другими VPC можно настроить пиринговое соединение.



Пример конфигурации схемы размещения:

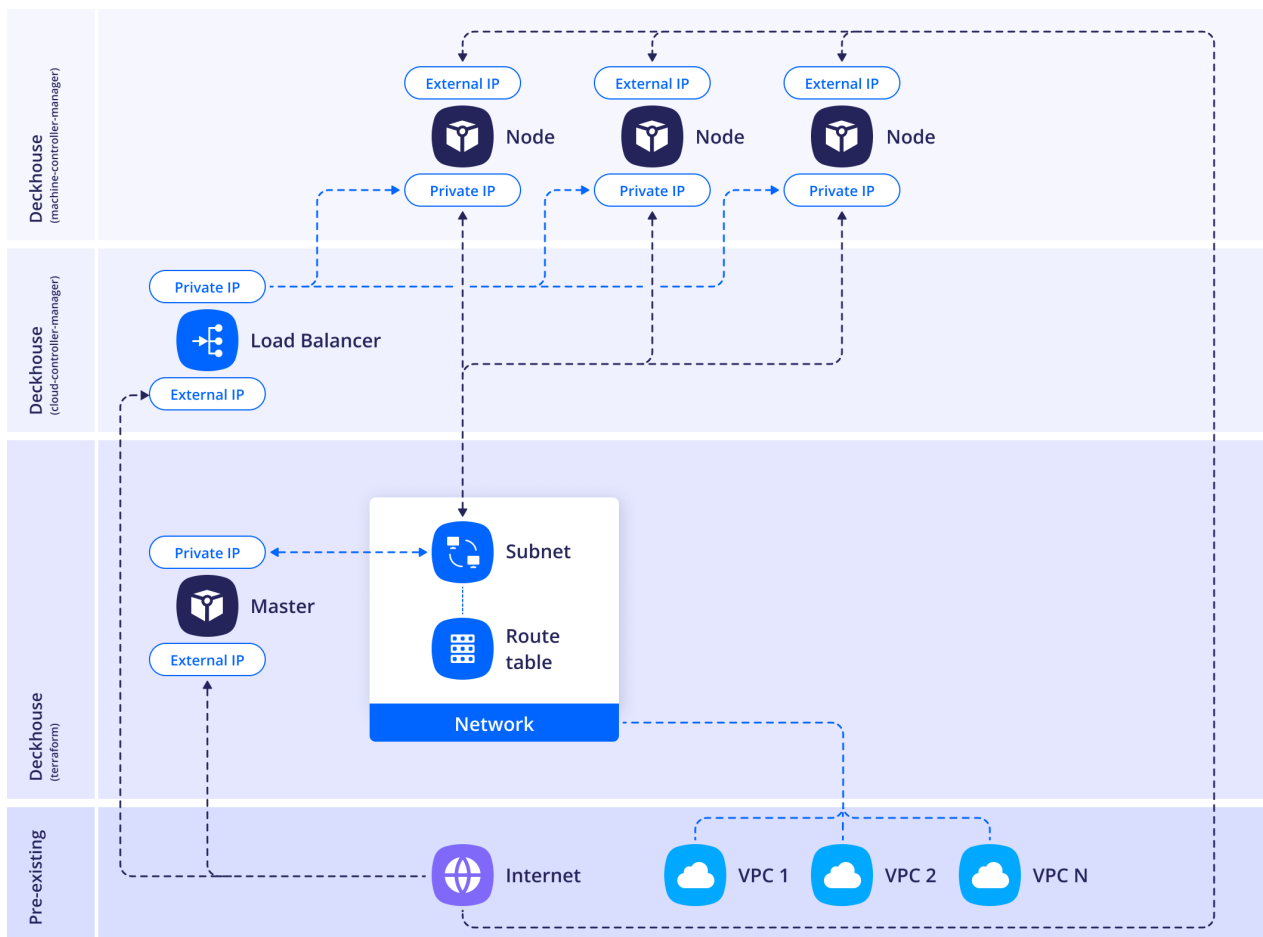
```

apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Необязательный параметр. Адреса из этого списка будут использованы для
  # адресации Cloud NAT.
  cloudNATAddresses:
    - example-address-1
    - example-address-2
  subnetworkCIDR: 10.0.0.0/24      # Обязательный параметр.
  # Необязательный параметр, список GCP VPC Networks, с которыми Kubernetes VPC
  # Network будет соединяться через пиринговое соединение.
  peeredVPCs:
    - default
  sshKey: "<SSH_PUBLIC_KEY>" # Обязательный параметр.
  labels:
    kube: example
  masterNodeGroup:
    replicas: 1
    zones:      # Необязательный параметр.
      - europe-west4-b
    instanceClass:
      machineType: n1-standard-4      # Обязательный параметр.
      image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
      diskSizeGb: 20                  # Необязательный параметр. Если не указан – используется локальный
диск.
      disableExternalIP: false        # Необязательный параметр, по умолчанию master-узел имеет
externalIP.
      additionalNetworkTags:          # Необязательный параметр.
        - tag1
      additionalLabels:               # Необязательный параметр.
        kube-node: master
  nodeGroups:
    - name: static
      replicas: 1
      zones:      # Необязательный параметр.
        - europe-west4-b
      instanceClass:
        machineType: n1-standard-4      # Обязательный параметр.
        image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
        diskSizeGb: 20                  # Необязательный параметр. Если не указан – используется локальный
диск.
        disableExternalIP: true         # Необязательный параметр, по умолчанию узлы не имеют externalIP.
        additionalNetworkTags:          # Необязательный параметр.
          - tag1
        additionalLabels:               # Необязательный параметр.
          kube-node: static
  provider:
    region: europe-west4              # Обязательный параметр.
    serviceAccountJSON: |             # Обязательный параметр.
      {
        "type": "service_account",
        "project_id": "sandbox",
        "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
        "private_key": "-----BEGIN PRIVATE KEY-----",
        "client_id": "342975658279248",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
      }

```

WithoutNAT

- Для кластера создается отдельная VPC, все узлы кластера имеют публичные IP-адреса.
- Между VPC кластера и другими VPC можно настроить пиринговое соединение.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24      # Обязательный параметр.
# Необязательный параметр, список GCP VPC Networks, с которыми Kubernetes VPC
# Network будет соединяться через пиринговое соединение.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                                # Необязательный параметр.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4      # Обязательный параметр.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
    diskSizeGb: 20                  # Необязательный параметр, Если не указан — используется локальный
диск.
    additionalNetworkTags:          # Необязательный параметр.
    - tag1
    additionalLabels:               # Необязательный параметр.
    kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                                # Необязательный параметр.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4      # Обязательный параметр.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
    diskSizeGb: 20                  # Необязательный параметр. Если не указан — используется локальный
диск.
    additionalNetworkTags:          # Необязательный параметр.
    - tag1
    additionalLabels:               # Необязательный параметр.
    kube-node: static
provider:
  region: europe-west4              # Обязательный параметр.
  serviceAccountJSON: |             # Обязательный параметр.
  {
    "type": "service_account",
    "project_id": "sandbox",
    "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
    "private_key": "-----BEGIN PRIVATE KEY-----",
    "client_id": "342975658279248",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
  }

```

Конфигурация

Интеграции с GCP осуществляется с помощью ресурса GCPClusterConfiguration, который описывает конфигурацию облачного кластера в GCP и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```



После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Пример конфигурации:

```

apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"

```

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference NodeGroup`). Инстанс-класс для cloud-провайдера GCP — это кастомный ресурс `GCPInstanceClass`, в котором указываются конкретные параметры самих машин.

Также для работы автоматически создаются `StorageClass`'ы, покрывающие все варианты дисков в GCP:

Тип	Репликация	Имя <code>StorageClass</code>
standard	none	pd-standard-not-replicated
standard	regional	pd-standard-replicated
balanced	none	pd-balanced-not-replicated
balanced	regional	pd-balanced-replicated
ssd	none	pd-ssd-not-replicated
ssd	regional	pd-ssd-replicated

Можно отфильтровать ненужные `StorageClass`'ы, для этого укажите их в параметре `exclude`.

Включение вложенной виртуализации

Для запуска виртуальных машин (например, KVM) внутри GCP-инстансов необходимо включить вложенную виртуализацию.

i Вложенная виртуализация поддерживается только на определённых типах машин. Список совместимых типов приведён в документации GCP.

```

apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  nestedVirtualization: true

```

Добавление дополнительных дисков

Чтобы подключить к инстансам дополнительные диски (например, для узлов хранилища LINSTOR, Ceph, NFS и аналогичных решений), задайте их в параметре `additionalDisks`:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
    - sizeGb: 200
      type: pd-ssd
    - sizeGb: 500
      type: pd-standard
  autoDelete: true
```

Настройка политик безопасности на узлах

На виртуальных машинах кластера в GCP может возникнуть необходимость ограничить или расширить входящий и исходящий трафик по различным причинам. Некоторые из них могут включать:

- Разрешение подключения к узлам кластера с виртуальных машин из другой подсети.
- Разрешение подключения к портам статического узла для работы приложения.
- Ограничение доступа к внешним ресурсам или другим виртуальным машинам в облаке по требованию службы безопасности.

Для всего этого необходимо применять дополнительные network tags.

Установка дополнительных network tags на статических и master-узлах

Данный параметр можно задать либо при создании кластера или в уже существующем кластере. В обоих случаях дополнительные network tags указываются в GCPClusterConfiguration:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalNetworkTags`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей соответствующую `nodeGroup`, в поле `additionalNetworkTags`.

Поле `additionalNetworkTags` содержит массив строк с именами network tags.

Установка дополнительных network tags на эфемерных узлах

Необходимо указать параметр `additionalNetworkTags` для всех ресурсов `GCPInstanceClass` в кластере, которым нужны дополнительные network tags.

Добавление CloudStatic узлов в кластер

К виртуальным машинам, которые вы хотите добавить к кластеру в качестве узлов, добавьте `Network Tag`, аналогичный префиксу кластера.

Префикс кластера можно узнать, воспользовавшись следующей командой:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
  | base64 -d | grep prefix
```

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-gcp`
- Кастомные ресурсы модуля `cloud-provider-gcp`
- Параметры модуля `cloud-provider-gcp`
- Кастомные ресурсы модуля `node-manager`

Частные облака

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства для интеграции с частными облаками, развёрнутыми на вашей инфраструктуре. Они позволяют автоматически использовать облачную инфраструктуру для заказа виртуальных машин и подключения их к кластеру, развёртыванию кластеров в облачных средах и выполнять масштабирование ресурсов в случае необходимости.

- VK Private Cloud;
- OpenStack;
- Huawei Cloud;
- Basis Dynamix;
- Deckhouse Virtualization Platform (DVP).

В следующих разделах описано, как настроить интеграцию с различными частными облаками.

Подключение и авторизация в VK Cloud

Для управления ресурсами в VK Private Cloud с помощью Deckhouse Kubernetes Platform необходимо подключиться к VK Private Cloud API.

Перечень API-сервисов VK Private Cloud, доступ к которым необходим для развёртывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к VK Private Cloud API, находятся в openrc-файле (OpenStack RC file).

Информация о получении openrc-файла с помощью стандартного веб-интерфейса VK Private Cloud и о способах его использования доступна в документации VK Private Cloud.

Если вы используете VK Private Cloud API cloud-провайдера, интерфейс получения openrc-файла может быть другим.

⚠️ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка VK Cloud

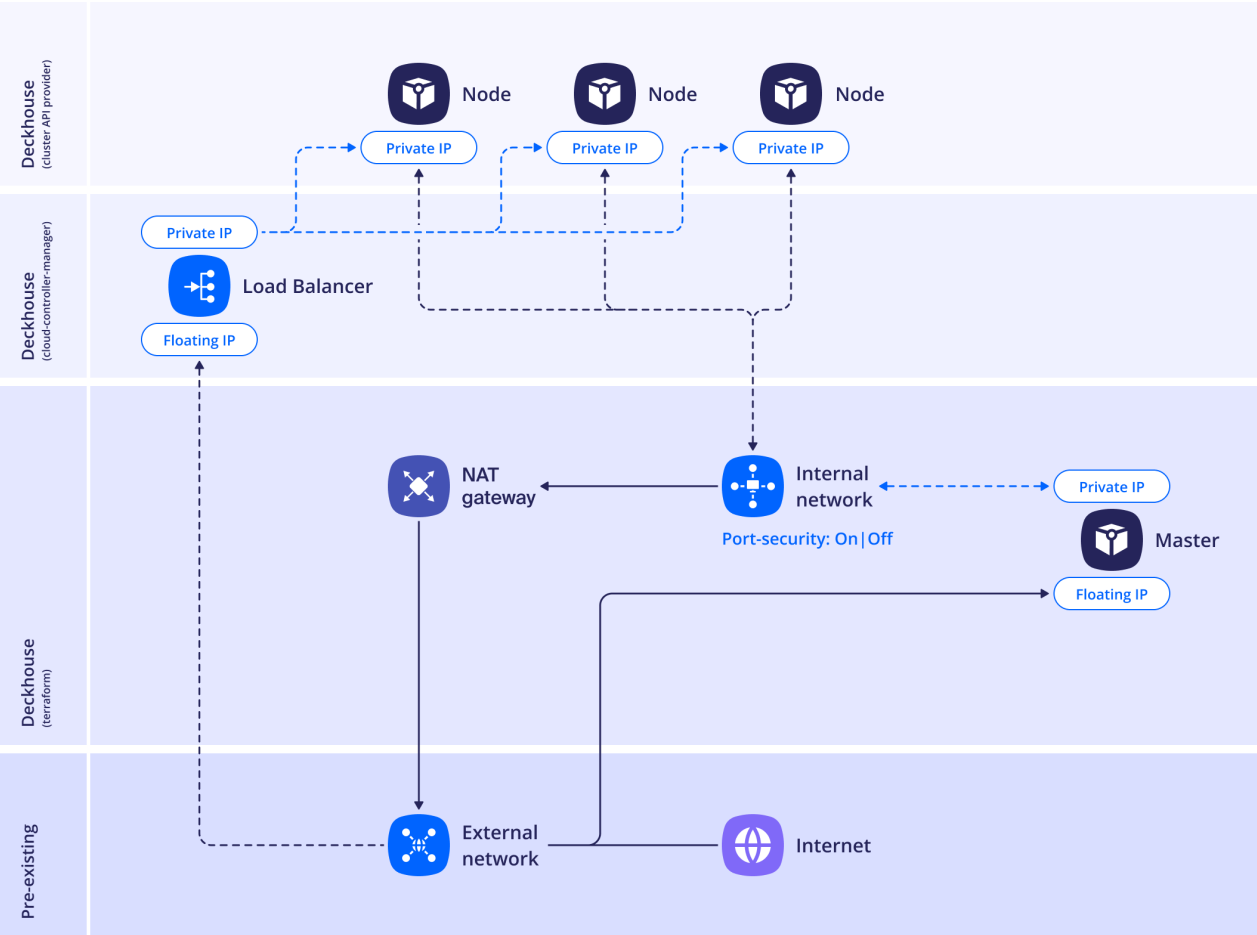
Схемы размещения

Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре VK Private Cloud и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.

⚠️ Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                 # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
```

```

internalNetworkSecurity: true|false      # Необязательный параметр, по умолчанию true.
externalNetworkName: shared              # Обязательный параметр.
bastion:
  zone: ru2-b                            # Необязательный параметр.
  volumeType: fast-ru-2b                  # Необязательный параметр.
  instanceClass:
    flavorName: m1.large                   # Обязательный параметр.
    imageName: ubuntu-20-04-cloud-amd64   # Обязательный параметр.
    rootDiskSize: 50                       # Обязательный параметр, по умолчанию 50 ГБ.
    additionalTags:
      severity: critical                   # Необязательный параметр.
      environment: production              # Необязательный параметр.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставленный поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                    # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default

```



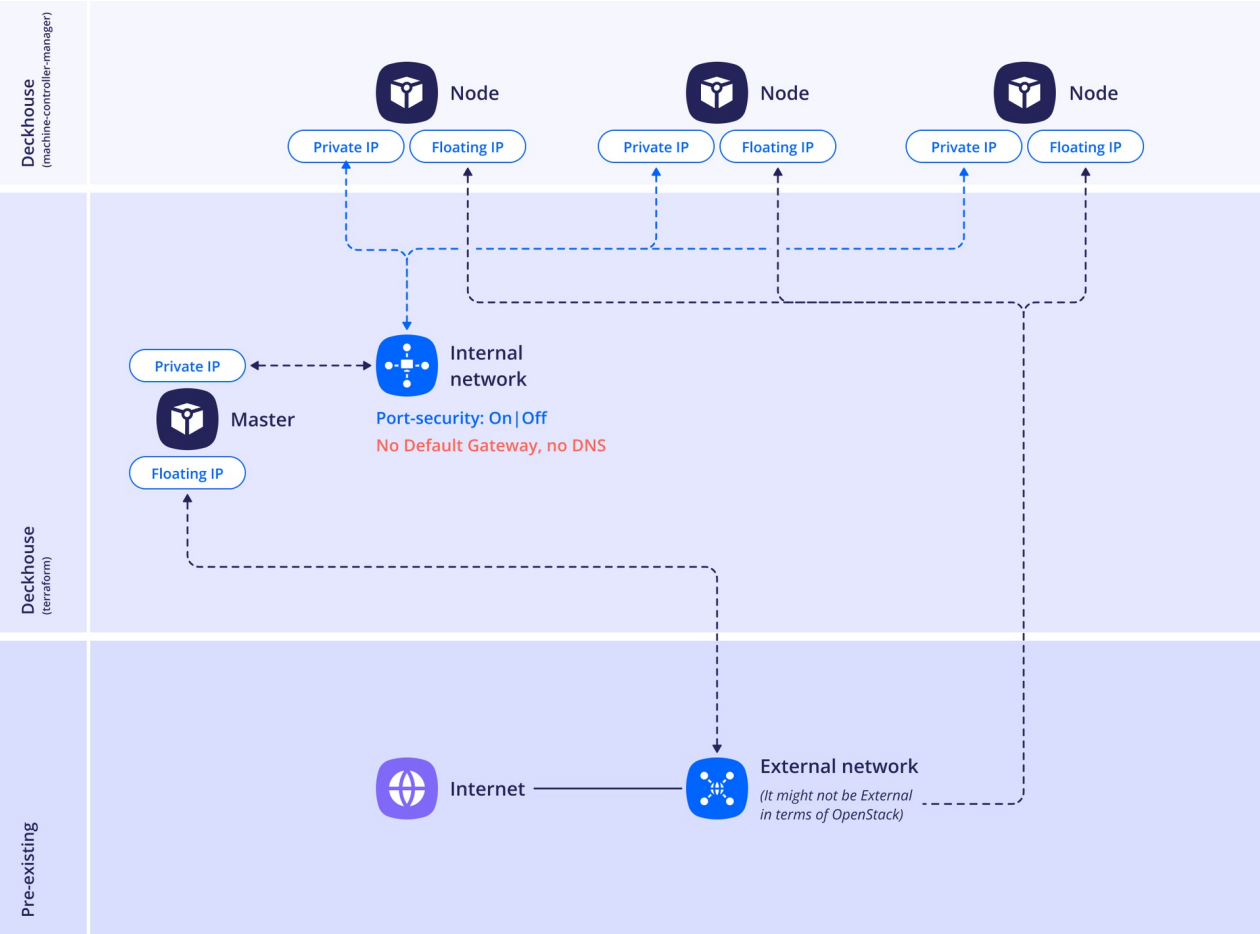
```
provider:
  ...
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

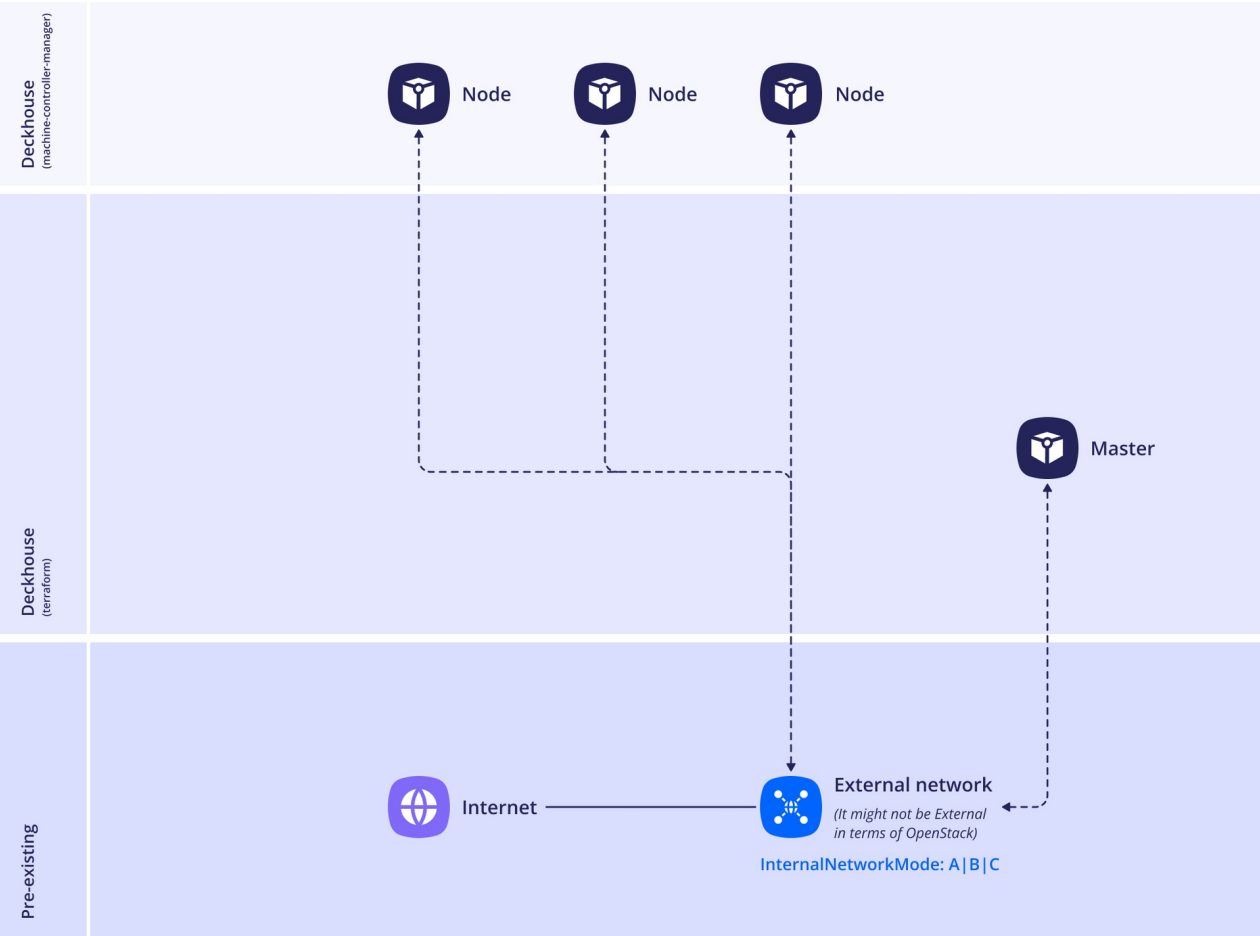
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                   # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

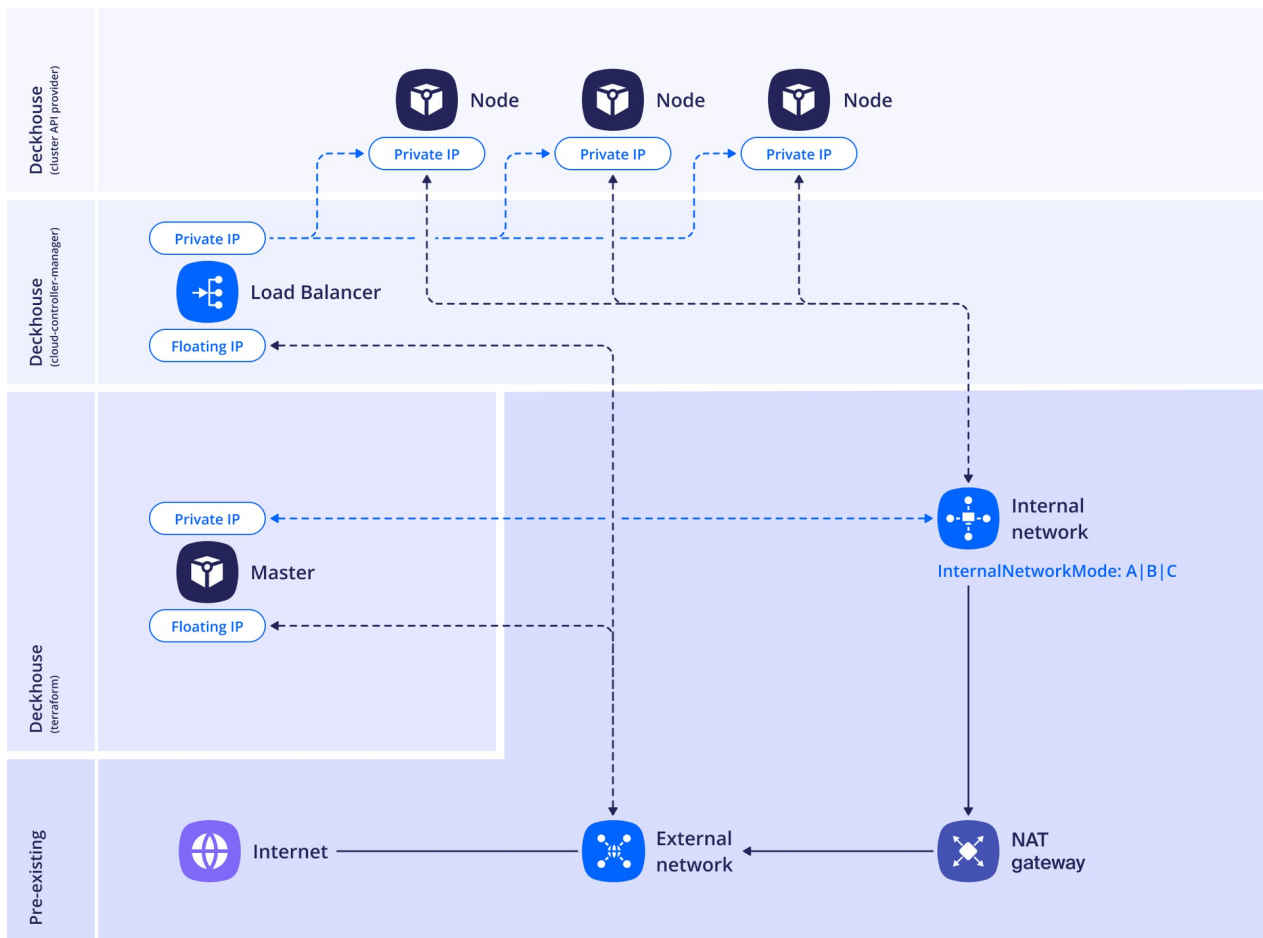
```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.



В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграции с VK Private Cloud осуществляется с помощью ресурса `OpenStackClusterConfiguration`, который описывает конфигурацию облачного кластера в VK Private Cloud и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера VK Private Cloud — это custom resource `OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».



При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов VK Private Cloud, необходимых для работы Deckhouse Kubernetes Platform в VK Private Cloud:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Адреса и порты API можно узнать в официальной документации.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим VM в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в VK Private Cloud

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте `openrc`-файл, который содержит `credentials` для обращения к API VK Private Cloud.

i Интерфейс получения `openrc`-файла может отличаться в зависимости от провайдера VK Private Cloud. Если провайдер предоставляет стандартный интерфейс для VK Private Cloud, скачать `openrc`-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него `openrc`-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из `openrc`-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

```
+-----+-----+-----+
| ID                                     | Name           | Is Public |
+-----+-----+-----+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | kol-high-iops  | True      |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops  | True      |
| 74101409-a462-4f03-872a-7de727a178b8 | kol-ssd        | True      |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd        | True      |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd            | True      |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1            | True      |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1            | True      |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph           | True      |
+-----+-----+-----+
```

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если VK Private Cloud не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:

```

+-----+-----+
|
|
|
+-----+-----+
|
|
+-----+-----+
|
|
+-----+-----+
| Field          | Value
|
+-----+-----+
|
|
+-----+-----+
| checksum        | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at      | 2020-01-10T07:23:48Z
|
| disk_format     | qcow2
|
| file            | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id              | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk        | 0
|
| min_ram         | 0
|
| name            | ubuntu-18-04-cloud-amd64
|
| owner           | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties      | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected       | False
|
| schema          | /v2/schemas/image
|
| size            | 343277568
|
| status          | active
|
| tags            |
|
| updated_at      | 2020-05-01T17:18:34Z
|
| virtual_size    | None
|
| visibility      | private
|
+-----+-----+
|
|
+-----+-----+
|
|
+-----+-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в VK Private Cloud

Диск узла может быть локальным или сетевым. В терминологии VK Private Cloud локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в VK Private Cloud flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

i При создании узлов с типом CloudEphemeral в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -C disk
```

Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ VK Private Cloud и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Подключение и авторизация в OpenStack

Для управления ресурсами в OpenStack с помощью Deckhouse Kubernetes Platform необходимо подключиться к OpenStack API.

Перечень API-сервисов OpenStack, доступ к которым необходим для развертывания, доступен в разделе Настройки.

Доступы пользователя, необходимые для подключения к OpenStack API, находятся в openrc-файле (OpenStack RC file).

Информация о получении openrc-файла с помощью стандартного веб-интерфейса OpenStack и о способах его использования доступна в документации OpenStack.

Если вы используете OpenStack API cloud-провайдера, интерфейс получения openrc-файла может быть другим.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Схемы размещения и настройка OpenStack

Схемы размещения

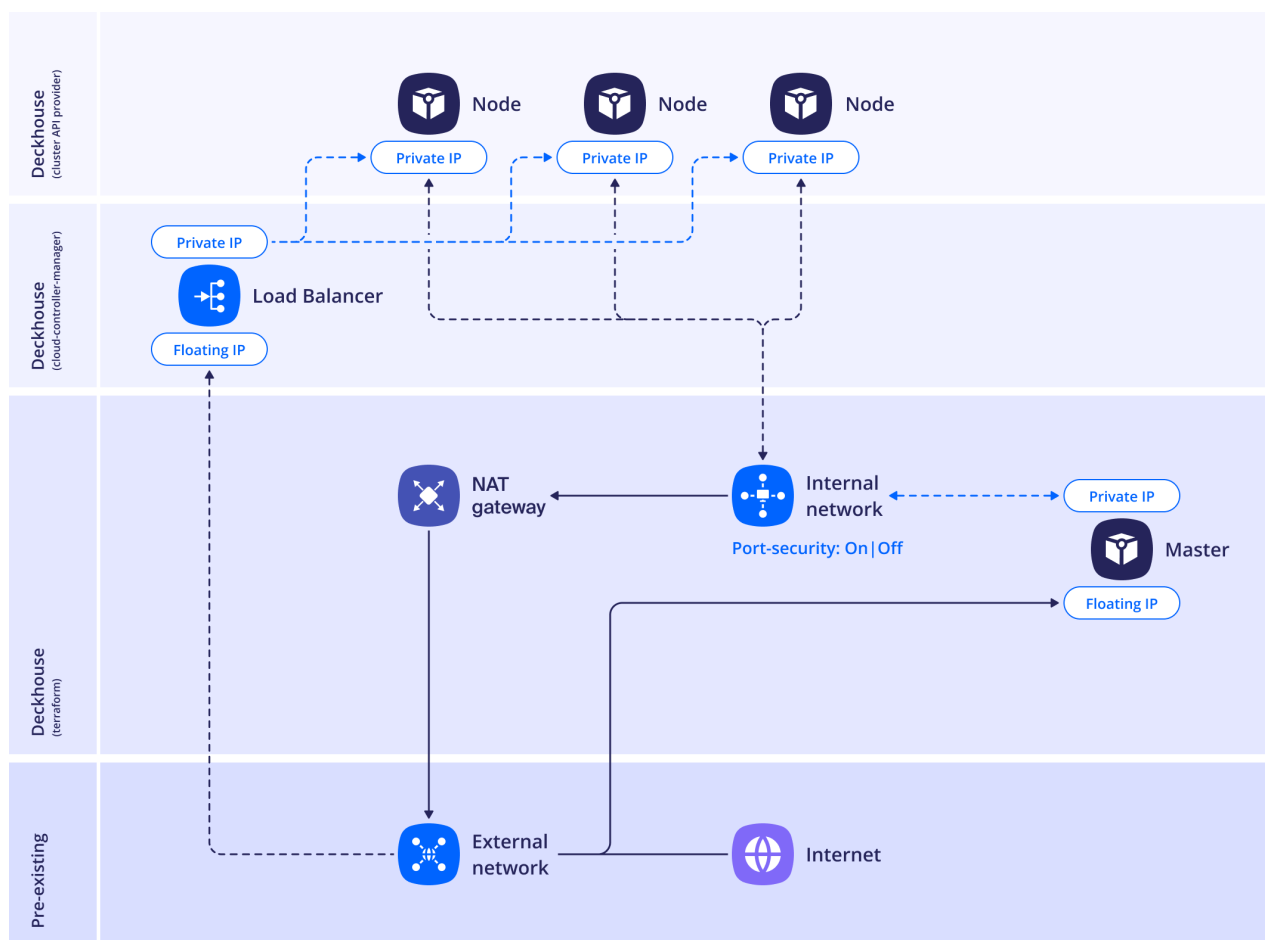
Данный раздел описывает возможные схемы размещения узлов кластера в инфраструктуре OpenStack и связанные с ними настройки. От выбора схемы (layout) зависят принципы сетевого взаимодействия, наличие публичных IP-адресов, маршрутизация исходящего трафика и способ подключения к узлам.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.



Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false        # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared               # Обязательный параметр.
  bastion:
    zone: ru2-b                             # Необязательный параметр.
    volumeType: fast-ru-2b                  # Необязательный параметр.
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-20-04-cloud-amd64  # Обязательный параметр.
      rootDiskSize: 50                     # Обязательный параметр, по умолчанию 50 ГБ.
      additionalTags:
        severity: critical                 # Необязательный параметр.
        environment: production           # Необязательный параметр.
  masterNodeGroup:
    replicas: 3
    instanceClass:
      flavorName: m1.large                  # Обязательный параметр.
      imageName: ubuntu-18-04-cloud-amd64  # Обязательный параметр.
      # Необязательный параметр. Если не указан – используется локальный диск.
      rootDiskSize: 50
      # Необязательный параметр, дополнительные группы безопасности.
      additionalSecurityGroups:
        - sec_group_1
        - sec_group_2
      additionalTags:
        severity: critical
        environment: production
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставленный поставщиком).
  volumeTypeMap:
```

```

# Если указан rootDiskSize, этот тип тома также будет
# использоваться для главного корневого тома.
ru-1a: fast-ru-1a
ru-1b: fast-ru-1b
ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIP Pools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...

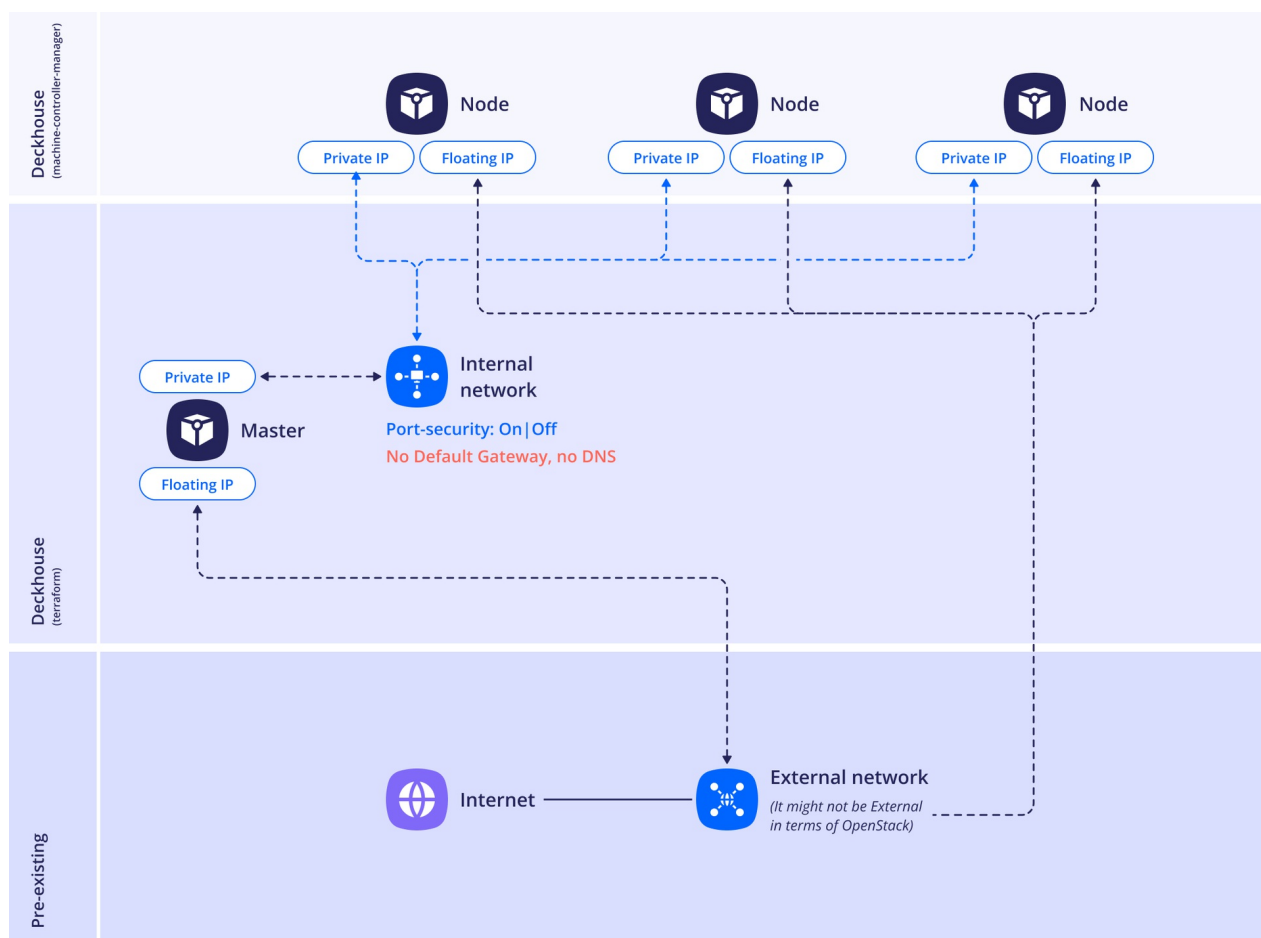
```

StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.



- В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.
- В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

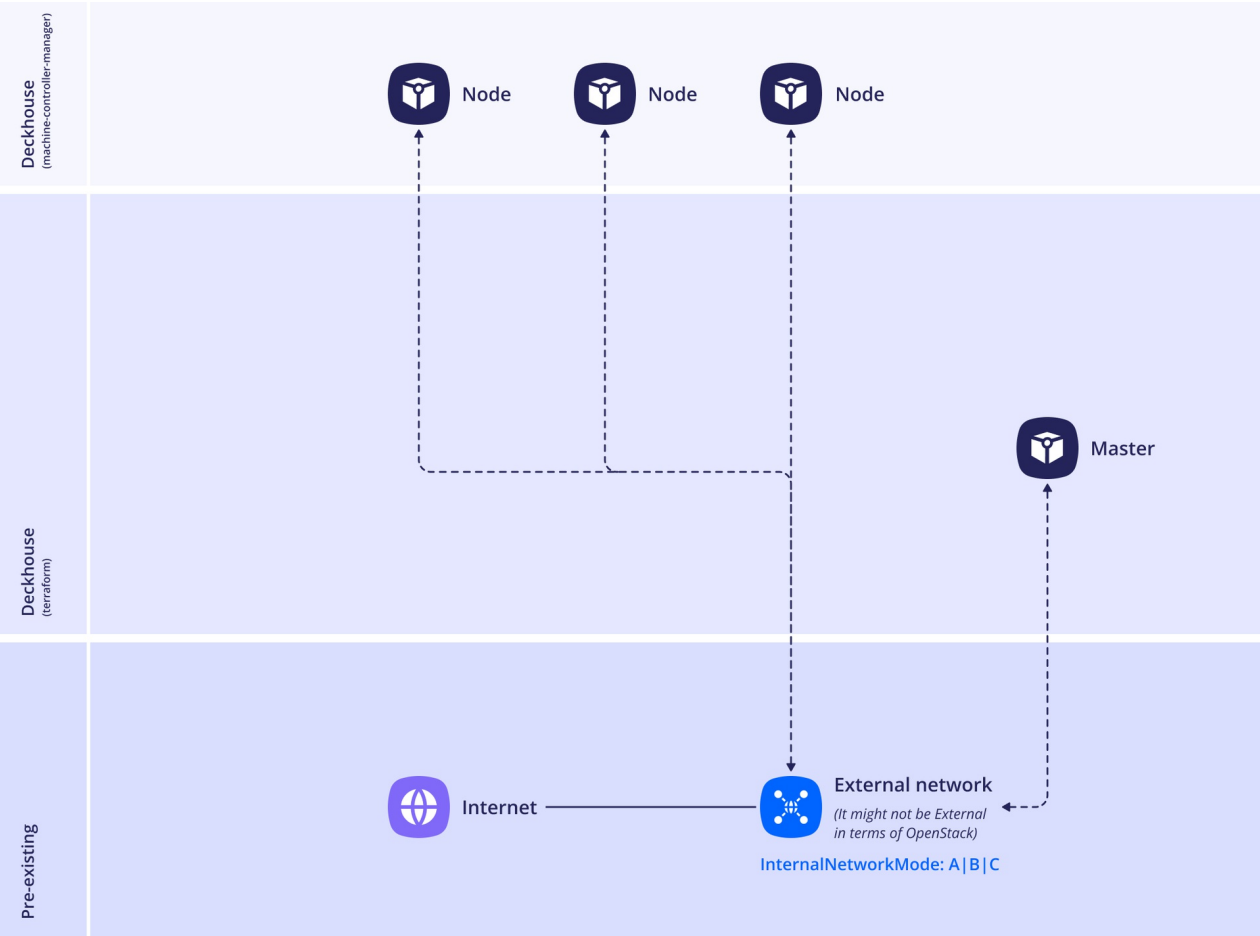
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                    # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

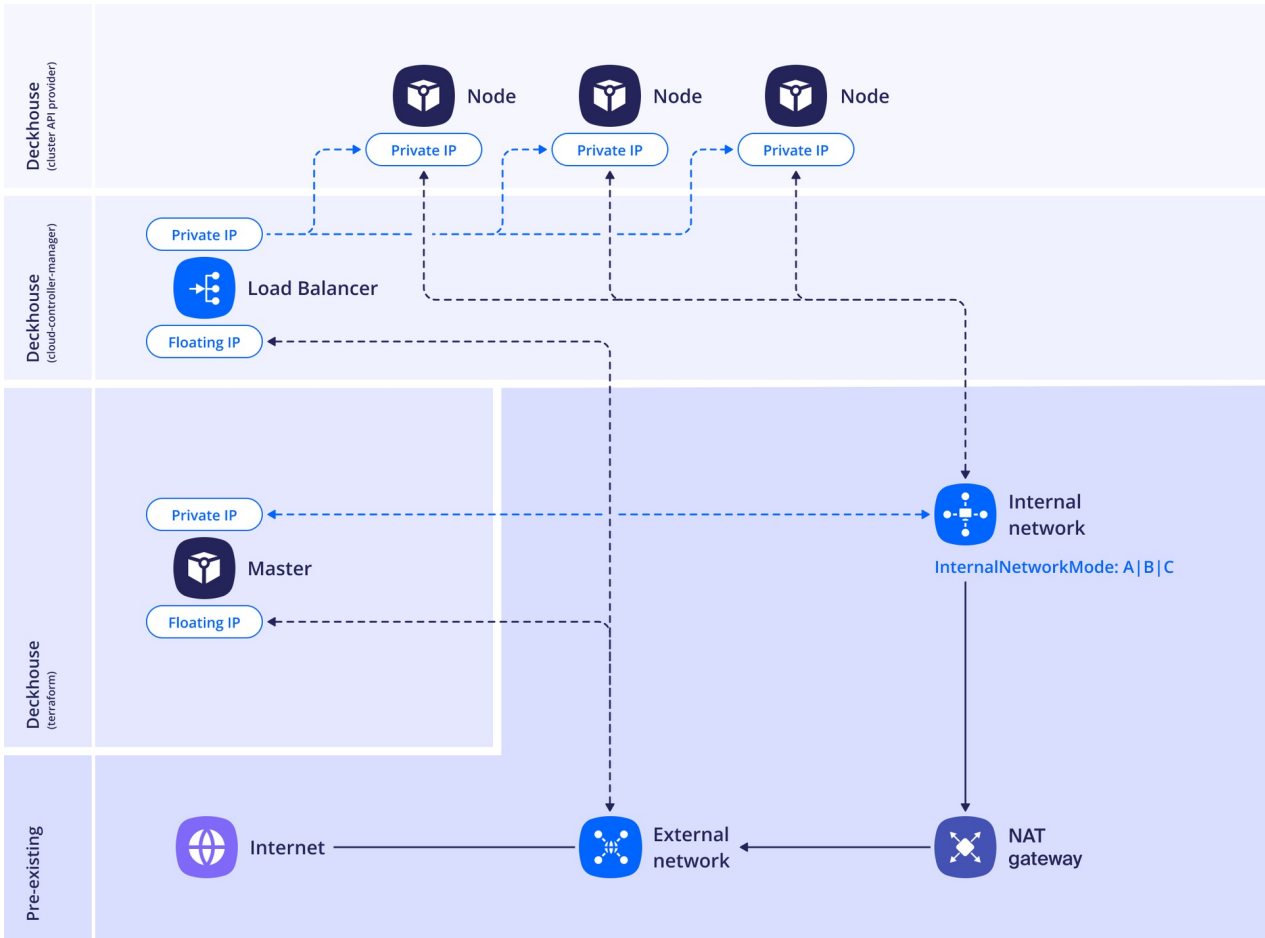
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

⚠ В данной схеме размещения не происходит управление SecurityGroups, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Конфигурация

Интеграции с OpenStack осуществляется с помощью ресурса `OpenStackClusterConfiguration`, который описывает конфигурацию облачного кластера в OpenStack и используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для облачного провайдера OpenStack — это `custom resource OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region`). Зоны других регионов использовать нельзя. Подробнее о настройке зон для узлов типов `CloudPermanent` и `CloudEphemeral` — в разделе «Как создать `NodeGroup` в зонах доступности?».



При изменении настроек модуля **пересоздания существующих объектов `Machines` в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`.

Примеры конфигурации

Ниже представлены два простых примера конфигурации облачного провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default
```

Список необходимых сервисов

Список сервисов OpenStack, необходимых для работы Deckhouse Kubernetes Platform в OpenStack:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Настройка LoadBalancer



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
```

Настройка и политики безопасности на узлах кластера

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим ВМ в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные групп безопасности.

Как загрузить образ в OpenStack

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте `openrc`-файл, который содержит `credentials` для обращения к API OpenStack.

i Интерфейс получения `openrc`-файла может отличаться в зависимости от провайдера OpenStack. Если провайдер предоставляет стандартный интерфейс для OpenStack, скачать `openrc`-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него `openrc`-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из `openrc`-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
openstack volume type list
```

Пример вывода:

```
+-----+-----+-----+
| ID                                     | Name           | Is Public |
+-----+-----+-----+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | kol-high-iops  | True      |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops  | True      |
| 74101409-a462-4f03-872a-7de727a178b8 | kol-ssd        | True      |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd        | True      |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd            | True      |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1            | True      |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1            | True      |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph           | True      |
+-----+-----+-----+
```

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если OpenStack не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
openstack image show ubuntu-18-04-cloud-amd64
```

Пример вывода:

```

+-----+-----+
|
|
|
|
+-----+-----+
|
|
|
|
+-----+-----+
|
|
|
|
+-----+-----+
| Field          | Value
|
|
+-----+-----+
|
|
|
|
+-----+-----+
| checksum        | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]'
|
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+-----+-----+
|
|
|
|
+-----+-----+

```

Проверка поддержки провайдером группы безопасности (security groups)

Выполните команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Настройка работы онлайн-изменений размера дисков

При успешном изменении размера диска с помощью OpenStack API, Cinder не передает информацию об этом изменении в Nova, в результате чего диск внутри гостевой операционной системы остается прежнего размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:


```
[nova]
interface = admin
insecure = {{ keystone_service_internalurl_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internalurl }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Более подробная информация в документации OpenStack-Ansible

Использование rootDiskSize

Диски в OpenStack

Диск узла может быть локальным или сетевым. В терминологии OpenStack локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Первый удаляется вместе с VM, а второй остается в облаке, когда VM удаляется.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все облачные провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Удаляется вместе с VM	Остается в облаке и может переиспользоваться
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр rootDiskSize

В OpenStackInstanceClass есть параметр `rootDiskSize`, и в OpenStack flavor есть параметр размера диска.

Какой диск будет заказан в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❗ <i>Необходимо задать размер. Без указания размера будет ошибка создания VM.</i>	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❗ Сетевой (rootDiskSize) и локальный (из flavor). Избегайте использования этого варианта, так как облачный провайдер будет взимать плату за оба диска.

i При создании узлов с типом CloudEphemeral в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в OpenStackInstanceClass.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию облачного провайдера.

Как проверить объем диска в flavor

Выполните следующую команду:

```
openstack flavor show m1.medium-50g -c disk
```

Пример вывода:

```
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию облачного провайдера

Если у облачного провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ OpenStack и загрузить его.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Офлайн-изменение размера диска

Если при изменении размера диска вы получаете следующую ошибку, необходимо уменьшить количество реплик StatefulSet до 0, подождать изменения размера дисков и вернуть обратно количество реплик, которое было до начала операции.

```
Warning VolumeResizeFailed 5s (x11 over 41s) external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Подключение и авторизация в Huawei Cloud

Требования

⚠️ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для корректной работы Deckhouse Kubernetes Platform (DKP) с Huawei Cloud необходимо:

- Убедиться, что на виртуальных машинах установлен пакет `cloud-init`.
- После запуска VM должны быть активны следующие службы:
 - `cloud-config.service`;
 - `cloud-final.service`;
 - `cloud-init.service`.

Доступ к Huawei Cloud API

DKP использует API Huawei Cloud для управления ресурсами. Для настройки доступа необходимо создать IAM-пользователя и предоставить ему необходимые права.

Создание группы пользователей

Для создания группы пользователей и назначения политик выполните шаги:

1. Перейдите в раздел «Identity and Access Management (IAM)».
2. Откройте вкладку «User Groups» и нажмите «Create User Group».
3. Укажите имя группы, например `deckhouse`, и нажмите «OK».
4. Выберите созданную группу и перейдите на вкладку «Permissions».
5. Нажмите «Authorize» и выберите следующие политики:
 - `ECS Admin`
 - `VPC Administrator`
 - `NAT Admin`
 - `ELB FullAccess`
 - `DEW KeypairFullAccess`
6. Подтвердите действия, нажав «Next», затем «OK», и завершите настройку кнопкой «Finish».

Создание IAM-пользователя

Для создания IAM-пользователя выполните шаги:

1. Перейдите во вкладку «Users» и нажмите «Create User».
2. Введите имя пользователя, например `deckhouse`.
3. В параметре «Access type» выберите `Programmatic access`, убедитесь, что `Management console access` отключён.

4. В параметре «Credential Type» выберите `Access key`.
5. Нажмите «Next», выберите ранее созданную группу, затем нажмите «Create».
6. Скачайте «Access Key ID» и «Secret Access Key». Эти данные необходимы для подключения к API Huawei Cloud и не могут быть восстановлены позже.

i Убедитесь, что сохраненные ключи надёжно защищены, так как они используются для подключения к API облака.

JSON-политики

Далее приведено содержание политик в формате JSON:

Политика «ECS Admin»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "ecs:*:*",
        "evs:*:get",
        "evs:*:list",
        "evs:volumes:create",
        "evs:volumes:delete",
        "evs:volumes:attach",
        "evs:volumes:detach",
        "evs:volumes:manage",
        "evs:volumes:update",
        "evs:volumes:use",
        "evs:volumes:uploadImage",
        "evs:snapshots:create",
        "vpc:*:get",
        "vpc:*:list",
        "vpc:networks:create",
        "vpc:networks:update",
        "vpc:subnets:update",
        "vpc:subnets:create",
        "vpc:ports:*",
        "vpc:routers:get",
        "vpc:routers:update",
        "vpc:securityGroups:*",
        "vpc:securityGroupRules:*",
        "vpc:floatingIps:*",
        "vpc:publicIps:*",
        "ims:images:create",
        "ims:images:delete",
        "ims:images:get",
        "ims:images:list",
        "ims:images:update",
        "ims:images:upload"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «VPC Administrator»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "vpc:vpcs:*",
        "vpc:routers:*",
        "vpc:networks:*",
        "vpc:subnets:*",
        "vpc:ports:*",
        "vpc:privateIps:*",
        "vpc:peerings:*",
        "vpc:routes:*",
        "vpc:lbaas:*",
        "vpc:vpns:*",
        "ecs:*:get",
        "ecs:*:list",
        "elb:*:get",
        "elb:*:list"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «NAT Admin»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "nat:*:*",
        "vpc:*:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «DEW KeypairFullAccess»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "kps:domainKeypairs:*",
        "ecs:serverKeypairs:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «ELB FullAccess»

```

{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "elb:*:*",
        "vpc:*:get*",
        "vpc:*:list*",
        "ecs:*:get*",
        "ecs:*:list*"
      ],
      "Effect": "Allow"
    }
  ]
}

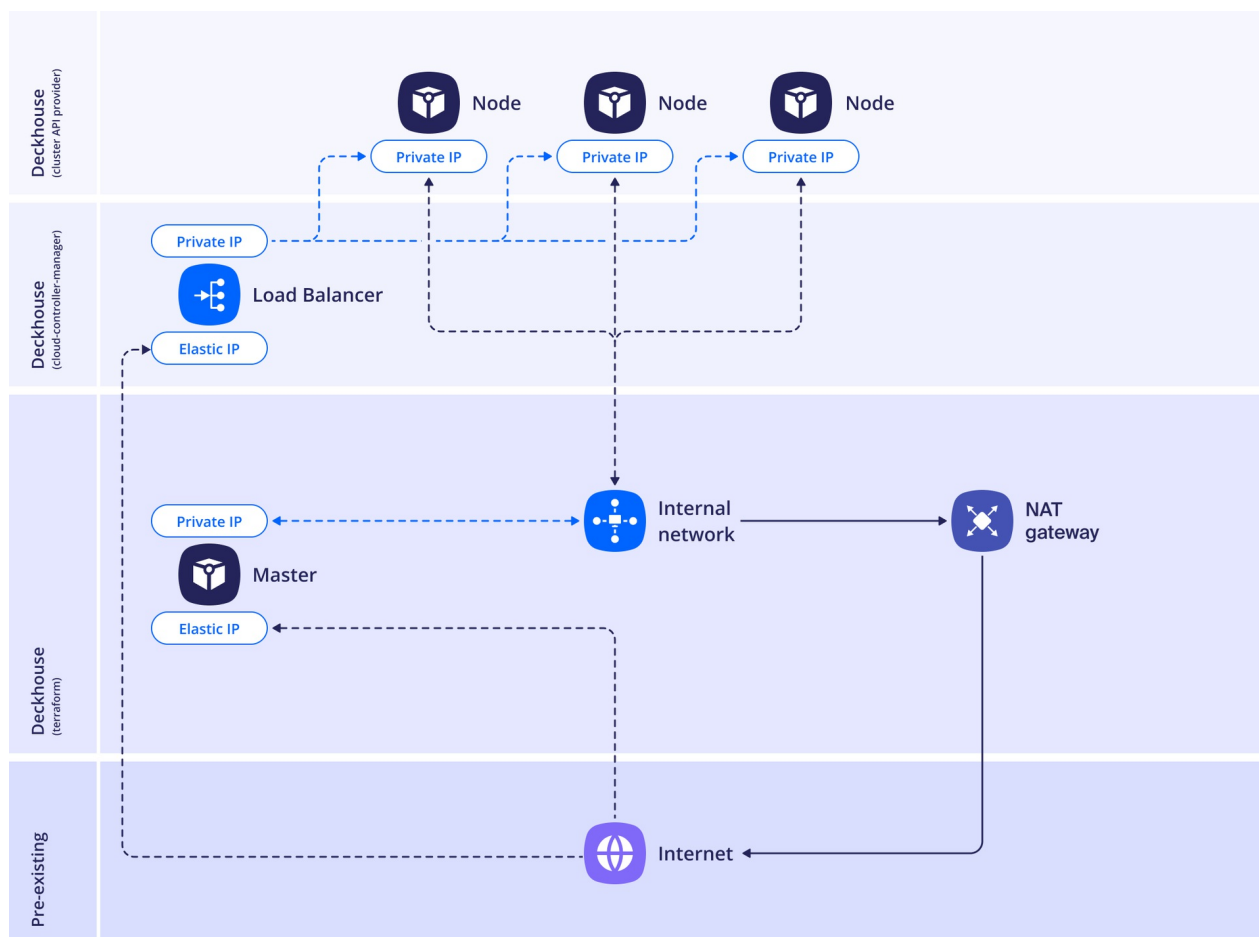
```

Схемы размещения и настройка Huawei Cloud

Данный раздел описывает схемы размещения кластера в инфраструктуре Huawei Cloud и связанные с ними параметры.

Standard

- Создаётся внутренняя сеть кластера со шлюзом к публичной сети.
- Master-узлу можно назначить Elastic IP.
- Узлы, управляемые Cluster API, не имеют публичных IP-адресов.

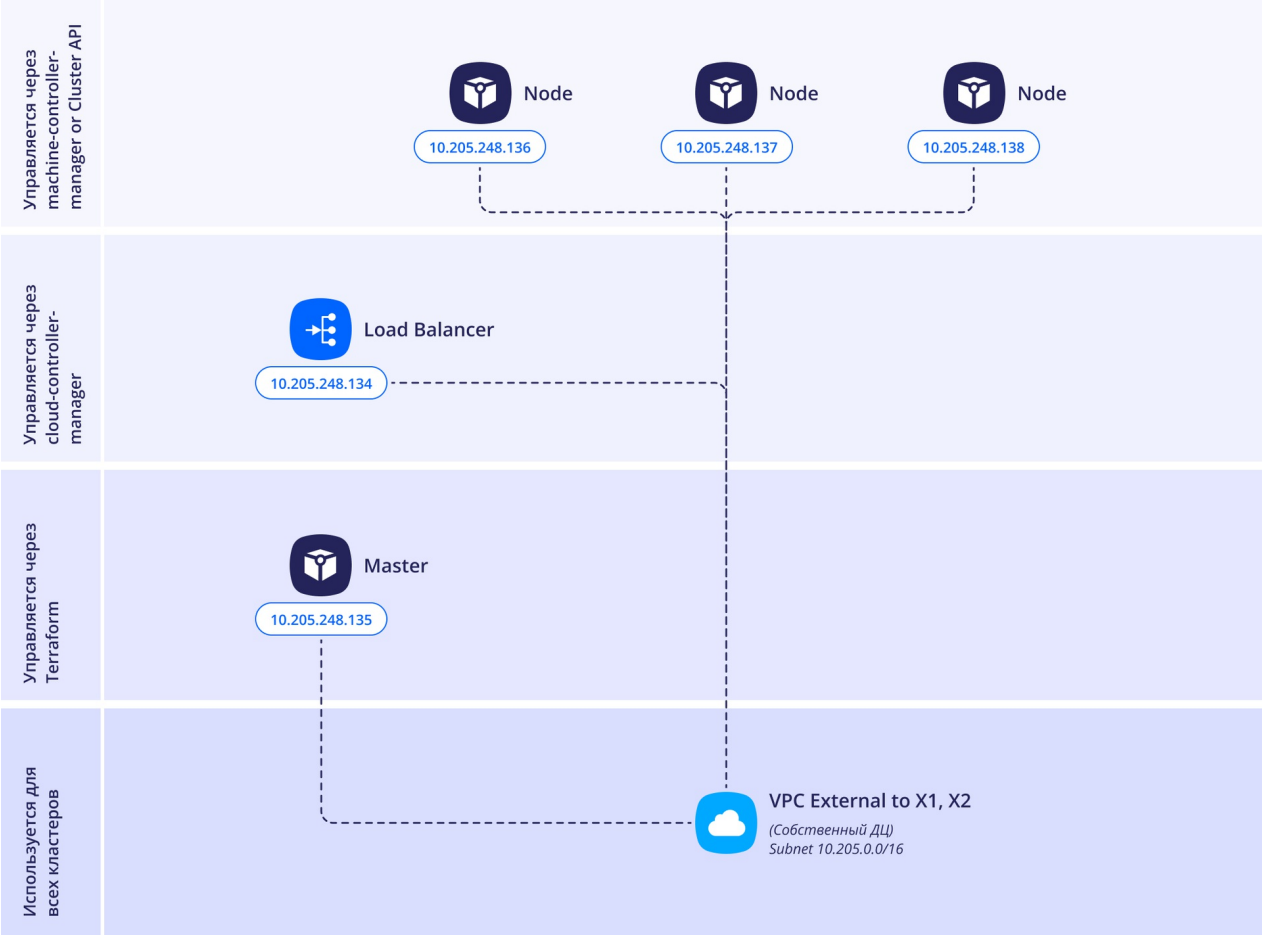


Пример конфигурации:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering

- Используется существующая сеть и подсеть VPC.
- Виртуальные машины подключаются к заданной подсети.



Пример конфигурации:

```

apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4

```

Хранилище и балансировка нагрузки в Huawei Cloud

Хранилище

Deckhouse Kubernetes Platform заказывает диски в Huawei Cloud с помощью CSI-драйвера. Для настройки типа хранилища используются параметры в объекте `HuaweiCloudClusterConfiguration`, в частности поле `volumeTypeMap`.

Пример настройки:

```

masterNodeGroup:
  volumeTypeMap:
    ru-moscow-1a: SSD

```

При указании параметра `rootDiskSize` будет использоваться тот же тип диска в качестве загрузочного (root) тома.

Также в `HuaweiCloudInstanceClass` можно задать:

- `rootDiskSize` — размер диска в Гиб;
- `rootDiskType` — тип корневого диска (например, SSD, GPSSD, SAS, и т.д.).

Пример:

```

spec:
  rootDiskSize: 50
  rootDiskType: GPSSD

```

Рекомендуется использовать наиболее быстрые типы дисков, доступные в регионе провайдера.

Балансировка нагрузки

Балансировка нагрузки в Huawei Cloud осуществляется с использованием сервисов ELB (Elastic Load Balancer). Для получения доступа к API балансировщика в IAM-политике пользователя необходимо разрешение `ELB FullAccess`.

В схеме размещения Standard доступен параметр `enableEIP`, который позволяет назначать Elastic IP для master-узлов.

По умолчанию узлы, управляемые Cluster API, не получают публичные IP-адреса.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-huaweicloud`
- Кастомные ресурсы модуля `cloud-provider-huaweicloud`

Интеграция со службами Huawei Cloud

Deckhouse Kubernetes Platform использует облачные возможности Huawei Cloud для работы Kubernetes-кластера. При этом поддерживаются следующие функции:

- управление ресурсами Huawei Cloud через `cloud-controller-manager`;
- заказ дисков с использованием CSI-драйвера;
- интеграция с модулем `node-manager`, позволяющая использовать `HuaweiCloudInstanceClass` в `NodeGroup`.

Работа с InstanceClass

Для описания параметров виртуальных машин используется ресурс `HuaweiCloudInstanceClass`. На него ссылается параметр `cloudInstances.classReference` `NodeGroup`.

Пример ресурса:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-huaweicloud`
- Кастомные ресурсы модуля `node-manager`

Подключение и авторизация в Basis Dynamix

i Интеграция с Basis Dynamix находится в экспериментальном режиме. Совместимость с будущими версиями не гарантируется, поведение может измениться.

Возможности интеграции включают:

- заказ и удаление виртуальных машин через API Basis Dynamix;
- использование подготовленных cloud-образов для развёртывания узлов;
- настройку параметров виртуальных машин (CPU, RAM, диск, сеть);
- поддержку внешней и внутренней сетей, включая настройку CIDR и DNS;
- работу с несколькими схемами размещения — с внешней или комбинированной сетью;
- управление хранением данных через pool и storage endpoint.

Требования

⚠ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для интеграции Deckhouse Kubernetes Platform (DKP) с Basis Dynamix необходимы:

- доступ к API-контроллеру и SSO-контур Basis Dynamix;
- данные учётной записи (account) и параметры приложения (`appId` и `appSecret`);
- размещённый в облаке cloud-образ операционной системы;
- внешняя сеть (`externalNetwork`) и, при необходимости, параметры внутренней сети (CIDR, DNS);
- сведения о хранилище (pool и storage endpoint);
- публичный SSH-ключ для доступа к узлам кластера.

Подготовка cloud-образа

Для развёртывания виртуальных машин DKP использует cloud-образы операционных систем, подготовленные вендорами для работы в виртуальной среде. Эти образы, как правило, включают:

- `cloud-init`;
- драйверы виртуального оборудования;
- гостевые агенты.

Рекомендуется использовать официальные cloud-образы, распространяемые в форматах `.img`, `.qcow2` и аналогичных.

Чтобы добавить образ в Basis Dynamix:

1. Перейдите в раздел «Образы» → «Шаблонные образы».
2. Загрузите подготовленный cloud-образ.

Настройка доступа к portalу

Если в вашей инфраструктуре отсутствует DNS-сервер, добавьте IP-адреса и домены портала Basis Dynamix вручную в шаблон `cloud-init`, чтобы обеспечить корректную работу системы при запуске узлов.

1. Отредактируйте шаблон файла `hosts`, расположенный в каталоге:

```
/etc/cloud/templates/
```

2. Добавьте соответствующие записи IP-адресов и доменных имен, характерные для вашей операционной системы и конфигурации. Это может включать записи для портала Basis Dynamix, такие как:

```
<IP-адрес> <доменное имя>
```

3. После изменений выполните очистку состояния `cloud-init`:


```
cloud-init clean
```

- 4. Создайте шаблон на основе подготовленной виртуальной машины. Это необходимо для корректной генерации файла `/etc/hosts` на этапе первого запуска.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-dynamix`
- Кастомные ресурсы модуля `cloud-provider-dynamix`

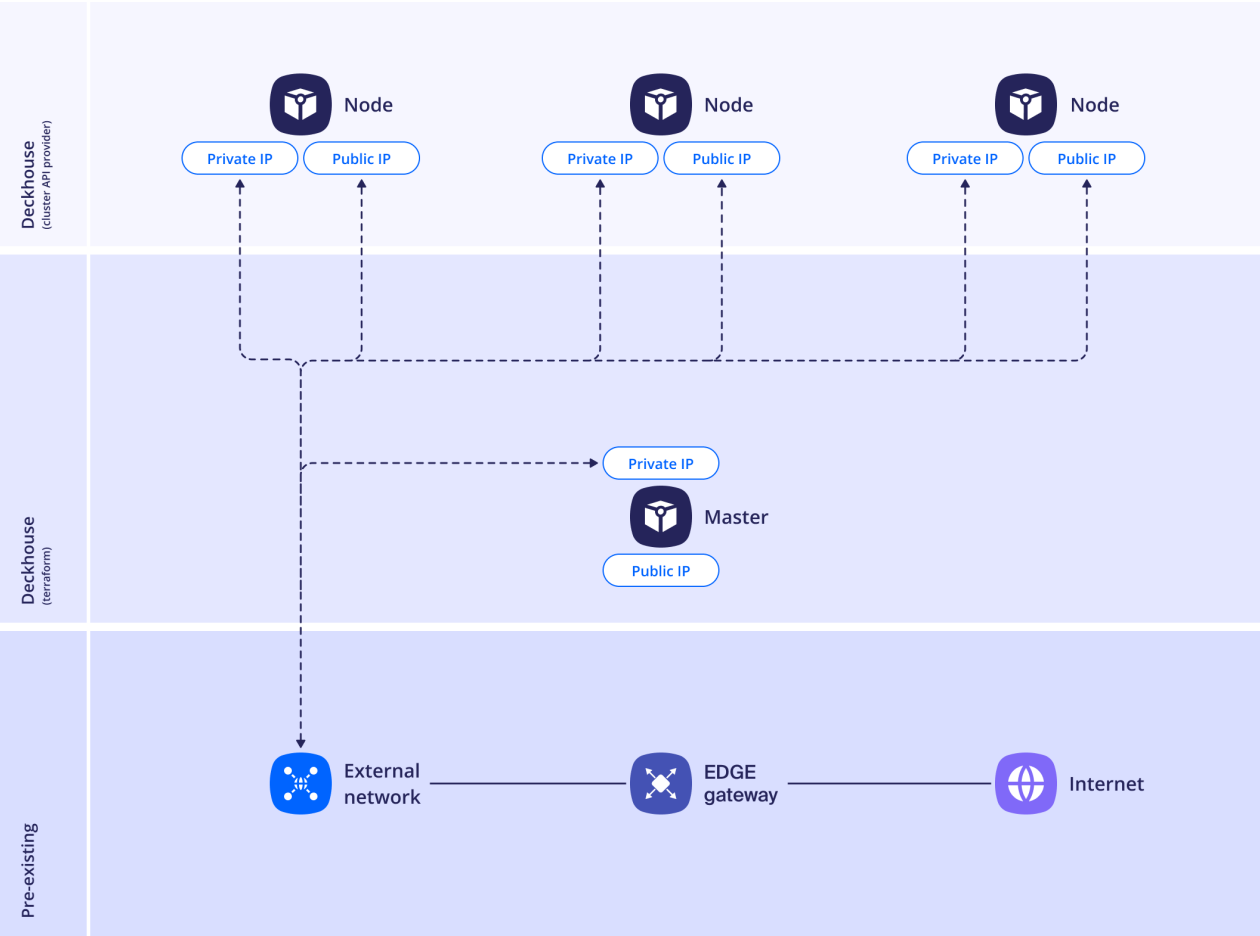
Схемы размещения и настройка Basis Dynamix

Deckhouse Kubernetes Platform поддерживает две схемы размещения в облаке Basis Dynamix:

- Standard — схема с использованием только внешней сети;
- StandardWithInternalNetwork — схема с внутренней (приватной) сетью и DNS-серверами.

Обе схемы позволяют управлять размещением узлов кластера, настройкой сетей, образом ОС и хранилищем данных.

Standard



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"

```

StandardWithInternalNetwork

Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
nodeNetworkCIDR: "10.241.32.0/24"
nameservers:
  - "10.0.0.10"
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"

```

Обязательные параметры

- `sshPublicKey` — публичный ключ для доступа к узлам;
- `location` — имя расположения облака (например, `dynamix`);
- `account` — имя аккаунта в облаке;
- `provider.controllerUrl`, `oAuth2Url`, `appId`, `appSecret` — параметры доступа к API;
- `imageName` — название образа ОС;
- `externalNetwork` — имя внешней сети;
- `storageEndpoint`, `pool` — параметры хранилища;
- `nodeNetworkCIDR` и `nameservers` — параметры внутренней сети (только для схемы `StandardWithInternalNetwork`).

После изменения параметров необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Хранилище и балансировка нагрузки в Basis Dynamix

Хранилище

Хранение данных в облаке Basis Dynamix осуществляется с использованием:

- `storageEndpoint` — имя хранилища, предоставленное провайдером;

- `pool` — имя пула хранения внутри указанного хранилища;
- `rootDiskSizeGb` — размер корневого диска каждой виртуальной машины (в гигабайтах).

Эти параметры задаются в секции `instanceClass` как для master-узлов, так и для рабочих групп узлов (`NodeGroup`).

Пример настройки:

```
masterNodeGroup:
  replicas: 1
  instanceClass:
    rootDiskSizeGb: 50
    storageEndpoint: SharedTatlin_G1_SEP
    pool: pool_a
```

i В текущей версии поддерживается только одно хранилище на группу узлов.

Балансировка нагрузки

Платформа Basis Dynamix не предоставляет встроенного балансировщика нагрузки. Для организации входящего трафика в кластер Deckhouse Kubernetes Platform рекомендуются следующие подходы:

1. Внешний балансировщик. Если в вашей инфраструктуре есть внешний балансировщик (аппаратный или программный), настройте проброс портов 80 и 443 на frontend-узлы кластера.
2. Использование MetalLB. Для обеспечения отказоустойчивой балансировки можно использовать MetalLB в L2-режиме.

Рекомендации:

- Выделите отдельную L2-сеть с DHCP и доступом в интернет.
- Настройте диапазон IP-адресов, из которого MetalLB будет анонсировать адреса.
- Обеспечьте подключение этой сети к frontend-узлам кластера.
- В конфигурации `VirtualMachine Template` оставьте сетевые интерфейсы пустыми — Deckhouse создаст их автоматически.

i Поддержка BGP-режима зависит от сетевой инфраструктуры и не гарантируется в Basis Dynamix.

Настройка Service типа LoadBalancer

Для настройки Service типа `LoadBalancer` добавьте в манифест Service следующие аннотации:

```
metadata:
  annotations:
    dynamix.cpi.flant.com/internal-network-name: <internal_name>
    dynamix.cpi.flant.com/external-network-name: <external_name>
```

Обе аннотации обязательны:

- `dynamix.cpi.flant.com/internal-network-name` — имя внутренней сети в Basis Dynamix;
- `dynamix.cpi.flant.com/external-network-name` — имя внешней сети в Basis Dynamix.

Термины «внутренняя сеть» и «внешняя сеть» используются в контексте Basis Dynamix. Внешняя сеть не обязательно должна быть публичной и может использовать серые IP-адреса.

Если одна из аннотаций не указана, `cloud-controller-manager` завершит обработку Service с ошибкой.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-dynamix`

Интеграция с облаком Basis Dynamix

Deckhouse Kubernetes Platform интегрируется с облачной платформой Basis Dynamix и использует ресурсы `DynamixInstanceClass` для описания характеристик виртуальных машин, разворачиваемых в кластере.

Основные возможности

- Заказ и удаление виртуальных машин через API Basis Dynamix;
- Настройка параметров виртуальных машин, включая количество CPU, объем памяти, размер корневого диска;
- Указание шаблона ОС (имя образа) и хранилища;
- Подключение к внешним сетям;
- Использование нескольких групп узлов с индивидуальными параметрами.

Пример описания `DynamixInstanceClass`:

```

apiVersion: deckhouse.io/v1
kind: DynamixInstanceClass
metadata:
  name: frontend
spec:
  numCPUs: 4
  memory: 8192
  rootDiskSizeGb: 40
  imageName: alt-p10-cloud-x86_64.img
  storageEndpoint: SharedTatlin_G1_SEP
  pool: pool_a
  externalNetwork: extnet_vlan_1700

```

На него ссылается параметр `cloudInstances.classReference NodeGroup`.

Рекомендации

- Размещайте образы ОС в разделе «Образы» → «Шаблонные образы» на портале Basis Dynamix.
- Используйте имена образов, точно соответствующие значениям `imageName` в `DynamixInstanceClass`.
- Убедитесь, что выбранное хранилище и пул доступны для всех узлов, размещаемых в кластере.
- Проверьте, что виртуальные машины имеют доступ в интернет и DNS-серверы.

Интеграция с облаком обеспечивает автоматическое масштабирование, настройку и управление узлами в соответствии с параметрами, заданными в `DynamixInstanceClass` и конфигурации кластера.

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-dynamix`
- Кастомные ресурсы модуля `node-manager`

Системы виртуализации

В Deckhouse Kubernetes Platform (DKP) поддерживаются следующие системы виртуализации:

- Deckhouse Virtualization Platform (DVP);
- VMware Cloud Director;
- VMware vSphere;
- zVirt.

В следующих разделах описано, как настроить интеграцию с различными системами виртуализации.

Интеграция с облаком Deckhouse Virtualization Platform

Deckhouse Kubernetes Platform интегрируется с инфраструктурой DVP и использует ресурсы `DVPInstanceClass` для описания характеристик виртуальных машин, создаваемых в составе кластера.

Основные возможности:

- Управление ресурсами DVP через модуль `cloud-controller-manager`;
- Заказ дисков с использованием компонента CSI storage;
- Интеграция с модулем `node-manager` для поддержки `DVPInstanceClass` при описании `NodeGroup`.



Модуль автоматически включается для всех облачных кластеров, развернутых в DVP. Модуль не имеет настроек.

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-dvp`
- Документация модуля `node-manager`

Подключение и авторизация в Deckhouse Virtualization Platform

Для взаимодействия с ресурсами в DVP компоненты Deckhouse Kubernetes Platform используют API DVP. Для настройки подключения создайте пользователя (`ServiceAccount`), назначьте ему соответствующие права доступа и сгенерируйте `kubeconfig`.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.



Если модуль `update-hostname` пакета `cloud-init` не отключён, рекомендуется изменить частоту его запуска с `always` на `once-per-instance`.

Для этого измените конфигурацию модуля `update-hostname` в файле `/etc/cloud/cloud.cfg`:

```
cloud_init_modules:
...
- [update-hostname, once-per-instance]
...
```

Также модуль `update-hostname` можно полностью отключить, удалив его из списка модулей `cloud_init_modules` в файле `/etc/cloud/cloud.cfg`.

Создание пользователя

Создайте нового пользователя в кластере DVP с помощью следующей команды:

```
d8 k create -f -<<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-demo
  namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: sa-demo-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: sa-demo
type: kubernetes.io/service-account-token
EOF
```

Добавление роли

Добавьте роль созданному пользователю в кластере DVP с помощью следующей команды:

```
d8 k create -f -<<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-demo-rb
  namespace: default
subjects:
- kind: ServiceAccount
  name: sa-demo
  namespace: default
roleRef:
  kind: ClusterRole
  name: d8:use:role:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Генерация kubeconfig

Сгенерируйте `kubeconfig`, который будет использоваться в файле первичной конфигурации кластера:

```

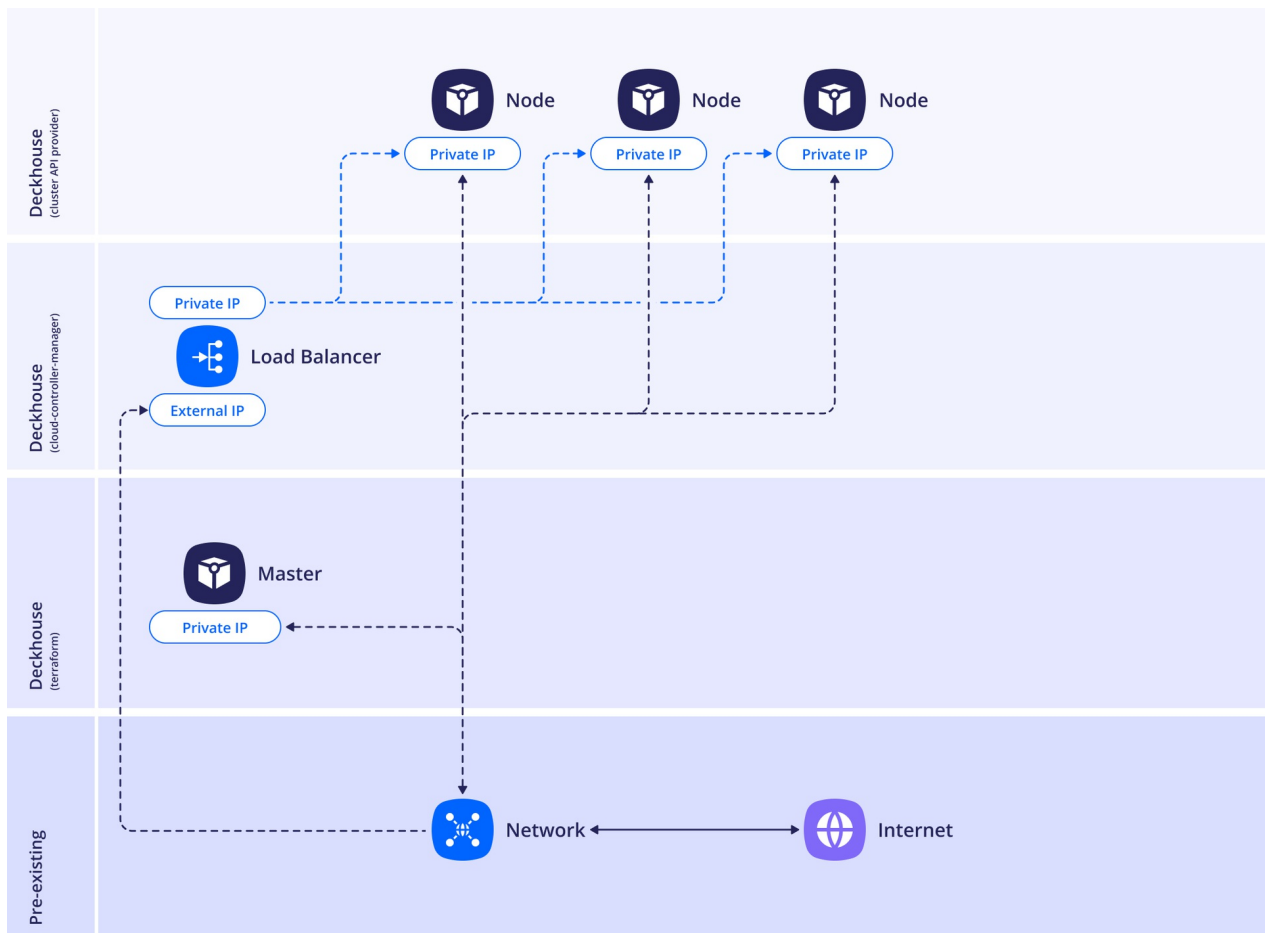
cat <<EOF > kubeconfig
apiVersion: v1
clusters:
- cluster:
    server: https://<KUBE-APISERVER-URL> # Замените на реальный адрес API-сервера кластера.
    name: <CLUSTER-NAME> # Замените на имя кластера.
contexts:
- context:
    cluster: <CLUSTER-NAME> # Замените на имя кластера.
    user: sa-demo
    namespace: default
    name: sa-demo-context
current-context: sa-demo-context
kind: Config
preferences: {}
users:
- name: sa-demo
  user:
    token: $(d8 k get secret sa-demo-token -n default -o json | jq -rC .data.token | base64 -d)
EOF

```

Закодируйте сгенерированный kubeconfig в кодировке Base64 (он указывается в файле первичной конфигурации в таком виде):

```
base64 kubeconfig | tr -d '\n'
```

Схемы размещения и настройка Deckhouse Virtualization Platform



Пример конфигурации схемы размещения:

```

---
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
masterNodeGroup:
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      ipAddresses:
        - Auto
      virtualMachineClassName: generic
    rootDisk:
      size: 50Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
    etcdDisk:
      size: 15Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
  provider:
    kubeconfigDataBase64: <KUBE_CONFIG>
    namespace: demo

```

Конфигурация

Deckhouse использует ресурс `DVPClusterConfiguration` для описания конфигурации кластера в DVP.

Для изменения конфигурации в работающем кластере выполните:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller edit provider-cluster-configuration
```

После изменения параметров узлов выполните команду:

```
dhctl converge
```

Пример конфигурации с множеством параметров:

```

apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- zone-a
- zone-b
- zone-c
region: r1
masterNodeGroup:
  replicas: 3
  zones:
  - zone-a
  - zone-b
  - zone-c
instanceClass:
  virtualMachine:
    cpu:
      cores: 1
      coreFraction: 100%
    memory:
      size: 4Gi
    virtualMachineClassName: generic
    ipAddresses:
    - 10.66.30.100
    - 10.66.30.101
    - 10.66.30.102
    additionalLabels:
      additional-vm-label: label-value
    additionalAnnotations:
      additional-vm-annotation: annotation-value
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
    nodeSelector:
      beta.kubernetes.io/os: linux
  rootDisk:
    size: 10Gi
    storageClass: linstor-thin-r1
    image:
      kind: ClusterVirtualImage
      name: ubuntu-2204
  etcdDisk:
    size: 10Gi
    storageClass: linstor-thin-r1
nodeGroups:
- name: worker
  zones:
  - zone-a
  - zone-b
  - zone-c
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      virtualMachineClassName: generic
    rootDisk:
      size: 10Gi
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
  provider:
    kubeconfigDataBase64: ZXhhbXBsZQo=
    namespace: default

```

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-dvp`

Подключение и авторизация в VMware Cloud Director

Подготовка ресурсов



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для управления ресурсами в VCD с помощью Deckhouse Kubernetes Platform необходимо настроить в системе следующие ресурсы:

- Organization
- VirtualDataCenter
- vApp (при схеме размещения `Standard`)
- StoragePolicy
- SizingPolicy
- Network (при схеме размещения `Standard`)
- EdgeRouter
- Catalog

Ресурсы Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter и Catalog должны быть предоставлены вашим поставщиком услуг VMware Cloud Director.

Network (внутренняя сеть) может быть настроена вашим поставщиком услуг VMware Cloud Director, либо вы можете настроить ее самостоятельно. При выборе схемы размещения `StandardWithNetwork` сеть создается автоматически. Далее описан способ самостоятельной настройки внутренней сети.

Права пользователя

Для доступа к API VMware Cloud Director достаточно одного из следующих вариантов:

- роль `Organization Administrator` с правом `Preserve All ExtraConfig Elements During OVF Import and Export`. Такой вариант не требует настройки отдельной роли, но предоставляет пользователю полный административный доступ к организации;
- пользовательская роль с набором прав, перечисленным ниже. Этого набора достаточно для создания кластера, заказа узлов, включая `CloudEphemeral` со статической адресацией и DHCP, а также для работы с CSI-дисками и `StorageClass` без предоставления дополнительных административных прав.

В VMware Cloud Director права пользователя складываются из роли и из **Rights Bundle** — набора прав, который публикуется для организации. Право `Preserve All ExtraConfig Elements During OVF Import and Export` нужно добавить и в роль, и в Rights Bundle организации.



Без права `Preserve All ExtraConfig Elements During OVF Import and Export` в роли и в Rights Bundle организации `userdata` не передаётся в виртуальные машины эфемерных узлов.

Чтобы создать пользовательскую роль, включите права, как показано ниже. Названия прав приведены как в интерфейсе VCD.

1. В разделах «ACCESS CONTROL» и «ADMINISTRATION» включите следующие права:
 - Organization → View: View Organization Administrative Details, View vApp ACL;
 - User → Manage: Manage user's own API token;
 - General → View: View Certificates Library, Administrator View, View Quota Policy Capabilities;
 - General → Manage: Administrator Control.

Name	View (47 / 121)	Manage (44 / 191)
ACCESS CONTROL		
Organization	View (27 / 7)	Manage (0 / 11)
	- Allow Access to All Organization VDCs - View Catalog ACL - View Access Control List of Organization VDCs ✓ View Organization Administrative Details - View Organization Associations - View OAuth Settings ✓ View vApp ACL	- Edit Organization Network Properties - Edit Access Control List of Organization VDCs - Edit Organization Associations - Edit Federation Settings - Edit Leases Policy - Edit Organization OAuth Settings - Edit Password Policy - Edit Organization Properties - Edit Quotas Policy - Edit SMTP Settings - Implicitly Import User/Group from IdP while Editing VDC ACL
Role		0 / 1
User	View (0 / 1)	Manage (1 / 3)
	View Users and Groups	✓ Manage user's own API token - Manage all users' API tokens - Manage Users and Groups
ADMINISTRATION		
General	View (17 / 12)	Manage (1 / 11)
	- View Advisories ✓ View Certificates Library - View General ACL ✓ Administrator View - View Error Details - View Shared Compute Policy ✓ View Quota Policy Capabilities - View Quota Policy - Test Connection - View task information - View Trusted Certificates - View vGPU Profile Consumption	- Manage Advisories - Manage Certificates Library - Manage General ACL ✓ Administrator Control - Send Notification - Publish Compute Policy - Migrate Storage - Manage Quota Policy - Resume / Cancel / Fail a Task - Update Task - Manage Trusted Certificates

2. В разделе «COMPUTE» → «Organization VDC» включите все права View и следующие права Manage:

- Edit Disk IOPS;
- Edit Tenant Kubernetes Policy;
- Change Owner;
- Create a Disk;
- Delete a Disk;
- Edit Disk Properties;
- Move a Disk;
- Create a Shared Disk.

В разделе «Provider VDC» → View включите **Limited Provider VDC Storage Policy View** и **Limited Provider VDC View**:

COMPUTE		
Container Applications	0 / 1	0 / 1
Kubernetes Namespace	0 / 2	0 / 1
Organization VDC	View (9 / 9)	Manage (7 / 19)
	✓ Limited Admin View ✓ View Compute Policies for an Organization VDC ✓ View Disk IOPS ✓ View Disk Encryption Status ✓ View Disk Properties ✓ View Organization VDC Storage Policy Capabilities ✓ View Organization VDC Extended Properties ✓ View Organization VDCs ✓ View Organization VDC Memory and CPU Reservation	✓ Edit Disk IOPS - Edit Tenant Kubernetes Policy ✓ Change Owner ✓ Create a Disk ✓ Delete a Disk ✓ Edit Disk Properties ✓ Move a Disk ✓ Create a Shared Disk - Edit Organization VDC Storage Policy - Disable Organization VDC Storage Policy - Remove Organization VDC Storage Policy - Set Default Storage Policy - Create Organization VDC - Delete Organization VDCs - Edit Organization VDC Name and Description - Enable / Disable Organization VDC - Edit Organization VDC Extended Properties - Manage Firewall - Edit VM-VM Affinity Rule
Provider VDC	View (2 / 2)	
	✓ Limited Provider VDC Storage Policy View ✓ Limited Provider VDC View	

3. В разделе «COMPUTE» → «vApp» включите права, как на скриншоте. Убедитесь, что включено **Preserve All ExtraConfig Elements During OVF Import and Export**. В проверенной роли среди View включены **View Encryption Status of VMs and VM's disks** и **View VM metrics**; **View vApp Shadow VMs** и остальные варианты Preserve ExtraConfig не включены:

Provider VDC	View (2/2)	
	✓ Limited Provider VDC Storage Policy View ✓ Limited Provider VDC View	
vApp	View (2/3)	Manage (23/29)
	- View vApp Shadow VMs ✓ View Encryption Status of VMs and VM's disks ✓ View VM metrics	- Change vApp Template Owner ✓ Preserve All ExtraConfig Elements During OVF Import and Export - Preserve Ethernet-Coalescing ExtraConfig Elements During OVF Import and Export - Preserve Latency ExtraConfig Elements During OVF Import and Export - Preserve ExtraConfig Elements During OVF Import and Export if they match patterns specified by the system administrator in the 'vapp.allowed.extra.config' configuration property - Preserve NUMA Node Affinity ExtraConfig Elements During OVF Import and Export - Change Owner ✓ Copy a vApp ✓ Create / Reconfigure a vApp ✓ Delete a vApp ✓ Download a vApp ✓ Edit vApp Properties ✓ Edit VM Compute Policy ✓ Edit VM CPU ✓ Edit VM CPU and Memory Reservation / Limit / Shares in all VDC types ✓ Edit VM Hard Disk ✓ Edit VM Memory ✓ Edit VM Network ✓ Edit VM Properties ✓ Enter / Exit vApp Maintenance Mode ✓ Manage VM Password Settings ✓ Start / Stop / Suspend / Reset a vApp ✓ Share a vApp ✓ Create / Revert / Remove a Snapshot ✓ Upload a vApp ✓ Access to VM Console ✓ Edit / View VM Boot Options ✓ Migrate / Force Undeploy / Relocate / Consolidate vApp VMs ✓ Allow metadata mapping domain to vCenter

4. В разделе «LIBRARIES» включите следующие права:

- Catalog → View: View Private and Shared Catalogs within Current Organization, View Shared Catalogs from Other Organizations (чтобы использовать образы из общих каталогов);
- Catalog → Manage: Add a vApp from My Cloud;
- Catalog Item → View: View vApp Templates / Media;
- Catalog Item → Manage: Copy / Move a vApp Template / Media, Edit vApp Template / Media Properties, Add to My Cloud.

В разделе «VMware Cloud Director Extension» → View включите View Tenant Portal Plugin Information:

VMware Cloud Director Extension	View (1/2)	Manage (0/1)
	- View Access Control List ✓ View Tenant Portal Plugin Information	Edit Access Control List
INFRASTRUCTURE		
Load Balancer	0/2	0/1
SDDC	0/1	0/4
vCenter		0/1
LIBRARIES		
Catalog	View (2/3)	Manage (1/8)
	- View Catalog vApp Shadow VMs ✓ View Private and Shared Catalogs within Current Organization ✓ View Shared Catalogs from Other Organizations	✓ Add a vApp from My Cloud - Change Owner - Allow External Publishing / Subscriptions for the Catalogs - Create / Delete a Catalog - Edit Catalog Properties - Share a Catalog to Other Organizations - Share a Catalog to Users / Groups within Current Organization - Allow VCSP Publish Subscribe Caching for Catalog
Catalog Item	View (1/1)	Manage (2/5)
	✓ View vApp Templates / Media	✓ Copy / Move a vApp Template / Media - Create / Upload a vApp Template / Media ✓ Edit vApp Template / Media Properties ✓ Add to My Cloud - Enable vApp Template / Media Download
NETWORKING		

5. В разделе «NETWORKING» включите следующие права:

- Edge Gateway → View: View Gateway;
- Edge Gateway Services → View: все пункты;
- Edge Gateway Services → Manage: Load Balancer Configure, NAT Configure;
- IP Spaces → View: View IP Spaces;
- IP Spaces → Manage: Allocate IP Spaces, Manage IP Spaces;
- Organization VDC Network → View: View Properties;
- Organization VDC Network → Manage: Edit Properties (чтобы создавать сети в организации).

NETWORKING		
> Distributed Firewall	☐ 0/1	☐ 0/2
> Edge Cluster	☐ 0/1	
▼ Edge Gateway	☒ View 1/1	☐ Manage 0/8
	✓ View Gateway	- Configure Syslog Server - Configure System Logging - Convert to Advanced Gateway - Create Gateway - Delete Gateway - Enable Distributed Routing - Update Gateway - Update Properties of Gateway
▼ Edge Gateway Services	☒ View 14/14	☐ Manage 2/15
	✓ BGP Routing View Only ✓ DHCP View Only ✓ Edge Gateway DNS View Only ✓ Firewall View Only ✓ IPSEC VPN View Only ✓ L2 VPN View Only ✓ Load Balancer View Only ✓ NAT View Only ✓ OSPF Routing View Only ✓ Remote Access View Only ✓ View Route Advertisement ✓ View SLAAC Profile ✓ SSL VPN View Only ✓ Static Routing View Only	- BGP Routing Configure - DHCP Configure - Edge Gateway DNS Configure - ECMP Routing Configure - Firewall Configure - IPSEC VPN Configure - L2 VPN Configure ✓ Load Balancer Configure ✓ NAT Configure - OSPF Routing Configure - Remote Access Configure - Configure Route Advertisement - Configure SLAAC Profile - SSL VPN Configure - Static Routing Configure
▼ IP Spaces	☒ View 1/1	☒ Manage 2/2
	✓ View IP Spaces	✓ Allocate IP Spaces ✓ Manage IP Spaces
▼ Organization VDC Network	☒ View 1/1	☐ Manage 1/3
	✓ View Properties	✓ Edit Properties - Import Network - Manage Manual IP Reservation

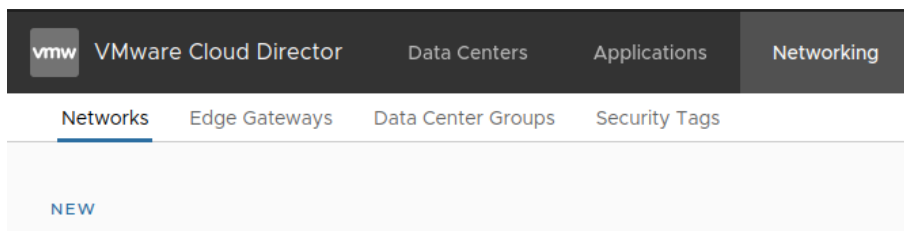
6. В разделе «Provider Gateway Services» включите все права View. В разделе «KubeClusterExtension» включите все права View и Manage:

> Network Pool	☐ 0/1	
▼ Provider Gateway Services	☒ View 6/6	☐ Manage 0/4
	✓ View Limited Provider Gateway BGP ✓ View Provider Gateway Firewall ✓ View Provider Gateway IPsec VPN ✓ View Provider Gateway NAT ✓ View Provider Gateway Routing (Simple View) ✓ View Limited Provider Gateway	- Manage Limited Provider Gateway BGP - Manage Provider Gateway Firewall - Manage Provider Gateway IPsec VPN - Manage Provider Gateway NAT
> Provider Network	☐ 0/1	
> Segment Profile Templates	☐ 0/1	
> VDC Group	☐ 0/1	☐ 0/3
OTHER		
> Annotation	☐ 0/2	☐ 0/3
> Branding UI Theme	☐ 0/2	☐ 0/3
> ContainerRegistryCredential	☐ 0/2	☐ 0/3
▼ KubeClusterExtension	☒ View 2/2	☒ Manage 3/3
	✓ Administrator View ✓ View	✓ Administrator Full Control ✓ Full Control ✓ Edit

Добавление сети

i Инструкция актуальна только для схемы размещения Standard.

1. Перейдите на вкладку «Networking» и нажмите «NEW»:



2. Выберите необходимый Data Center:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Scope

Organization Virtual Data Center

Provides connectivity for VMs in the selected VDC only

	Name	Allocation Model	Organization
☒	flant-df-lab_DC1	Pay-As-You-Go	flant-df-lab_DC1

1 - 1 of 1 data center(s)

CANCEL

NEXT

3. На этапе «Network type» выберите «Routed»:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Network Type

Select the type of network that you are about to create

☒ Routed

This type of network provides controlled access to machines and networks outside of the VDC or VDC Group through an edge gateway.

☐ Isolated

This type of network provides a fully isolated environment, which is accessible only by this organization VDC or VDC Group.

CANCEL

PREVIOUS

NEXT

4. Присоедините EdgeRouter к сети:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Edge Connection

Name	External Networks	Org VDC Networks
EDGE-flant-df-lab_DC1	1	2

1 - 1 of 1 Edge Gateway(s)

Distributed Routing

Guest VLAN Allowed

CANCEL

PREVIOUS

NEXT

5. Укажите имя сети и CIDR:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

General

Name *

internal

Description

Dual-Stack Mode

Gateway CIDR *

192.168.199.1/24

Shared

CANCEL

PREVIOUS

NEXT

6. Не добавляйте «Static IP Pools», поскольку будет использоваться DHCP:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Static IP Pools

Gateway CIDR192.168.199.1/24 ⓘ

Static IP Pools

Enter an IP range (format: 192.168.1.2 - 192.168.1.100)

ADD

MODIFY

REMOVE

Total IP addresses: 0

CANCEL

PREVIOUS

NEXT

7. Укажите адреса DNS-серверов:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

DNS

Primary DNS8.8.8.8

Secondary DNS8.8.4.4

DNS suffix

CANCEL

PREVIOUS

NEXT

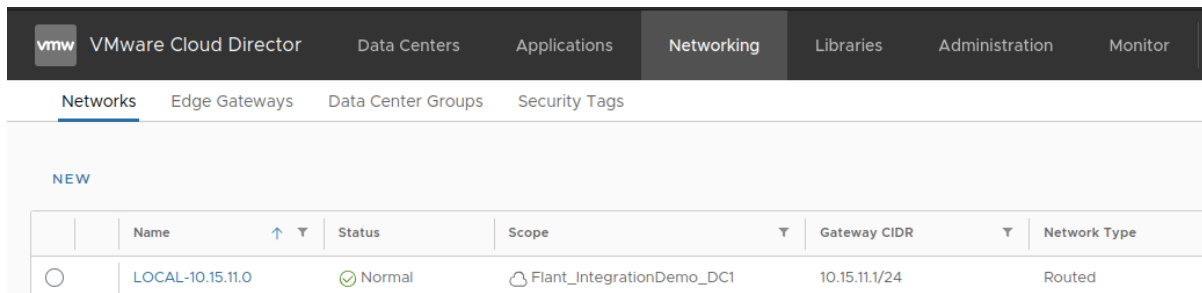
Настройка DHCP

i Инструкция актуальна только для схемы размещения **Standard**.

Для динамического заказа узлов включите DHCP-сервер для внутренней сети.

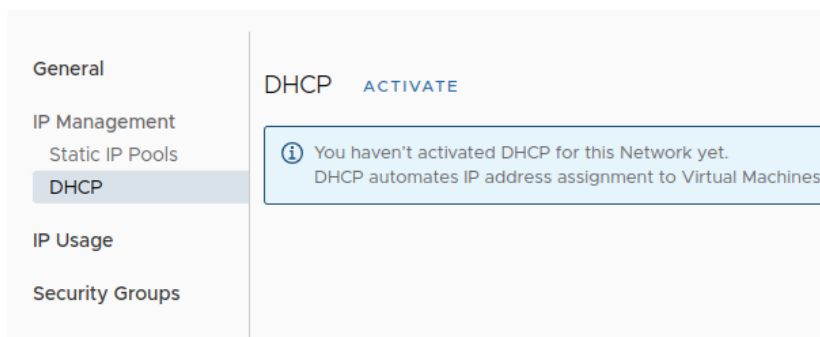
i Рекомендуем выделить начало диапазона адресов сети на системные нагрузки (control plane, frontend-узлы, системные узлы), а остальное выделить на DHCP-пул. Например, для сети по маске /24 будет достаточно выделения 20 адресов под системные нагрузки.

1. Перейдите на вкладку «Networking» и откройте созданную сеть:



VMware Cloud Director						
	Data Centers	Applications	Networking	Libraries	Administration	Monitor
Networks Edge Gateways Data Center Groups Security Tags						
NEW						
	Name	Status	Scope	Gateway CIDR	Network Type	
☐	LOCAL-10.15.11.0	Normal	Flant_IntegrationDemo_DC1	10.15.11.1/24	Routed	

2. В открывшемся окне выберите пункт «IP Management» → «DHCP» → «Activate»:



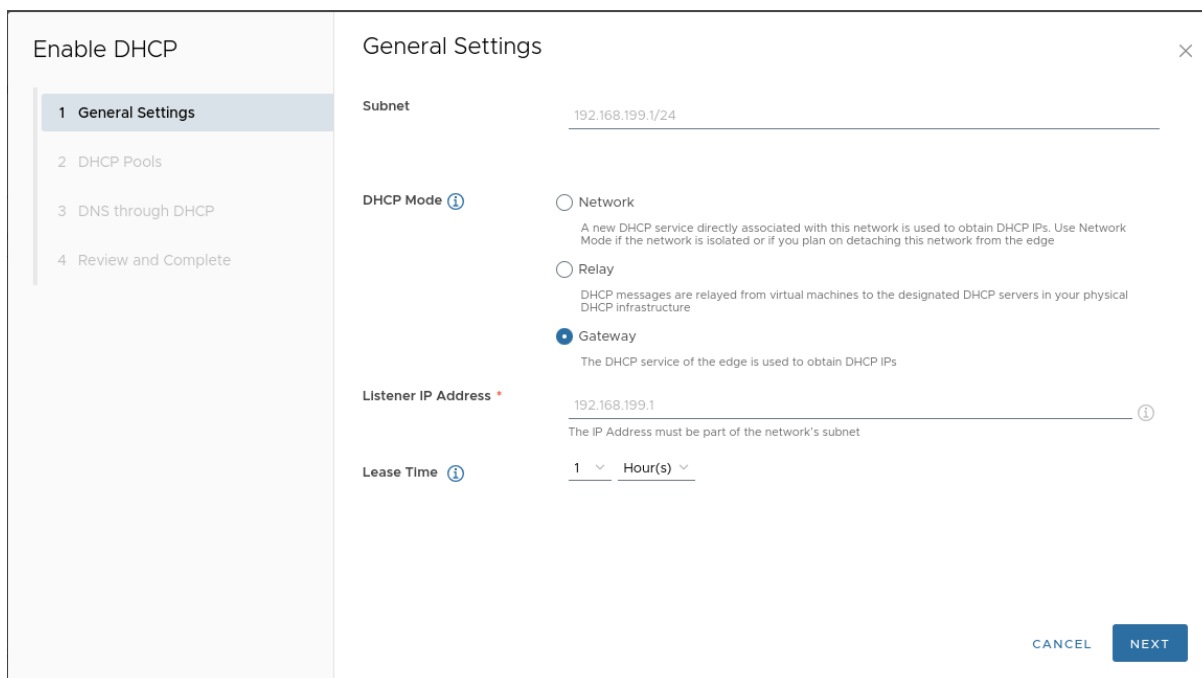
General
IP Management
Static IP Pools
DHCP
IP Usage
Security Groups

DHCP

ACTIVATE

i You haven't activated DHCP for this Network yet.
DHCP automates IP address assignment to Virtual Machines

3. На вкладке «General settings» настройте параметры аналогично примеру:



Enable DHCP

- 1 General Settings**
- 2 DHCP Pools
- 3 DNS through DHCP
- 4 Review and Complete

General Settings

Subnet: 192.168.199.1/24

DHCP Mode

☐ Network
A new DHCP service directly associated with this network is used to obtain DHCP IPs. Use Network Mode if the network is isolated or if you plan on detaching this network from the edge

☐ Relay
DHCP messages are relayed from virtual machines to the designated DHCP servers in your physical DHCP infrastructure

☒ **Gateway**
The DHCP service of the edge is used to obtain DHCP IPs

Listener IP Address: 192.168.199.1
The IP Address must be part of the network's subnet

Lease Time: 1 Hour(s)

CANCEL **NEXT**

4. Добавьте пул:

Enable DHCP

- General Settings
- DHCP Pools**
- DNS through DHCP
- Review and Complete

DHCP Pools

ADD

Pools
192.168.199.20-192.168.199.254

1 item(s)

Total IP addresses: 235

CANCEL
PREVIOUS
NEXT

5. Укажите адреса DNS-серверов:

Enable DHCP

- General Settings
- DHCP Pools
- DNS through DHCP**
- Review and Complete

DNS through DHCP

When a VM is connected to a network using DHCP mode, it will obtain these DNS IPs from the DHCP service. You can add up to two DNS servers using IPv4 or IPv6 addresses for both.

DNS Server 1
8.8.8.8

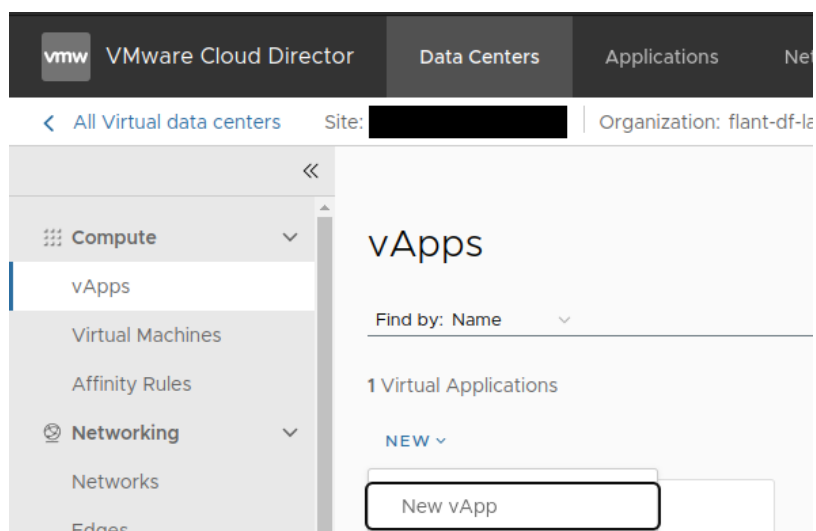
DNS Server 2
1.1.1.1

CANCEL
PREVIOUS
NEXT

Добавление vApp

i Инструкция актуальна только для схемы размещения Standard.

1. Перейдите на вкладку «Data Centers» → «vApps» → «NEW» → «New vApp»:



2. Укажите имя и включите vApp:

New vApp

Name *

deckhouse

Description

Power on

☒

Virtual Machines	OS	Compute

ADD VIRTUAL MACHINE

CANCEL

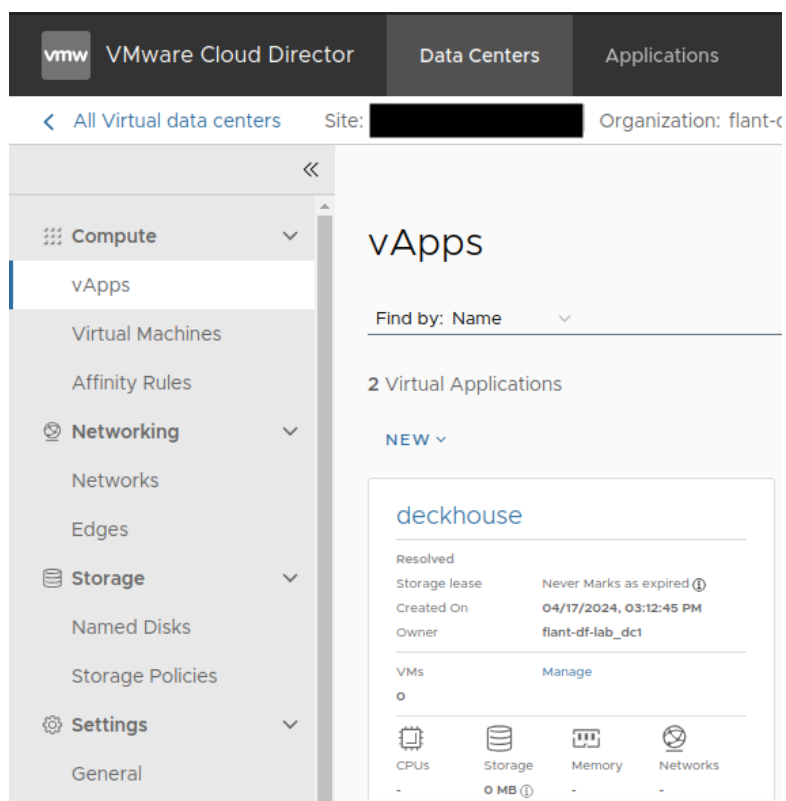
CREATE

Добавление сети к vApp

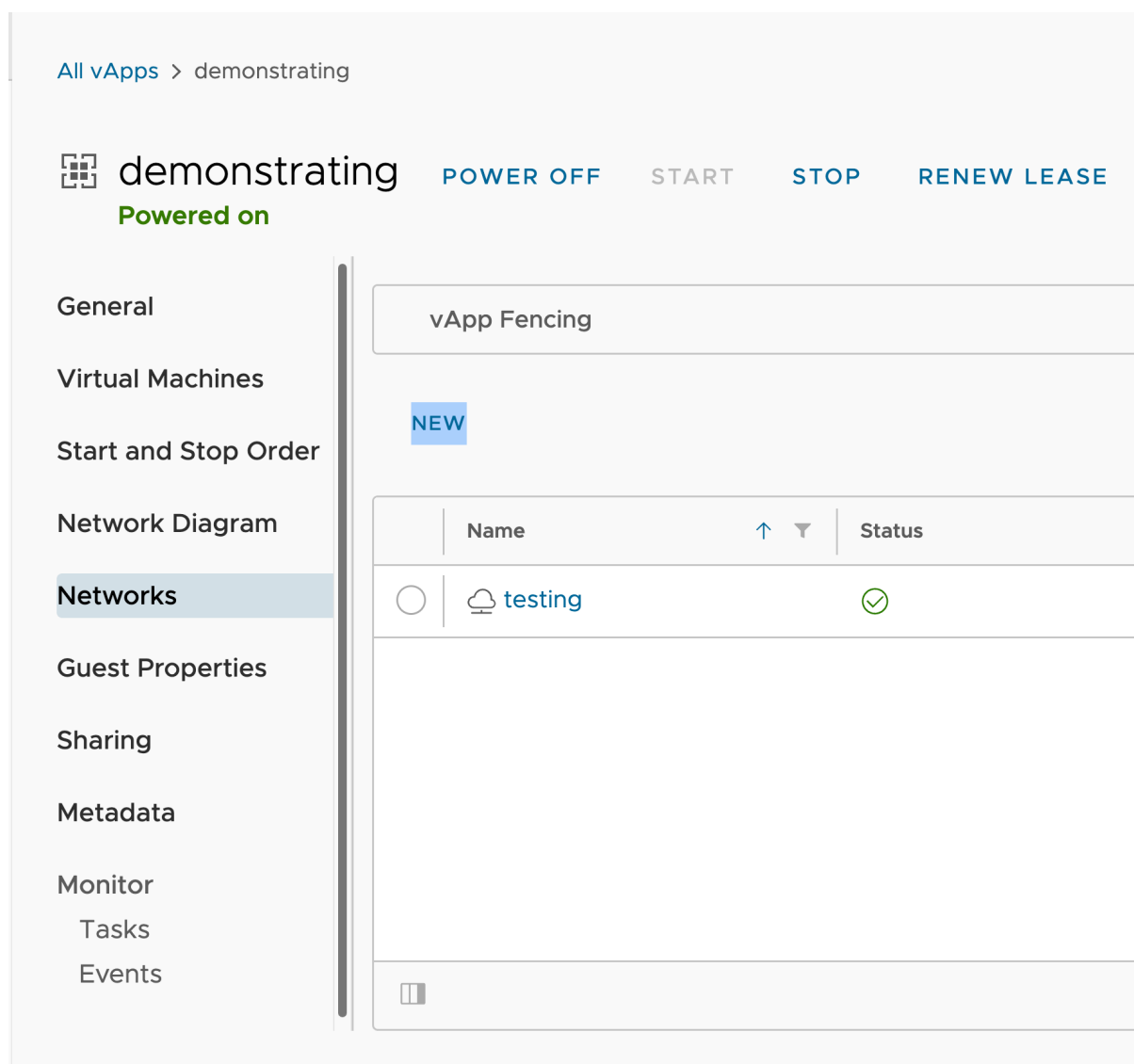
i Инструкция актуальна только для схемы размещения Standard.

После создания vApp присоедините к ней созданную внутреннюю сеть.

1. Перейдите на вкладку «Data Centers» → «vApps» и откройте необходимый vApp:



2. Перейдите на вкладку «Networks» и нажмите «NEW»:



3. В появившемся окне выберите тип «Direct» и выберите сеть:

Add Network to deckhouse

Type

☒ Direct

☐ Isolated

Org VDC Network Connection

	Name	Status	Organization VDC	Gateway CIDR	Network Type	Connected To	IP Pool Consumed	Shared	Route Advertised
☒	internal		flant-df-la...	192.168.199.1/24	Routed	-	-		-

CANCEL

ADD

Входящий трафик

Входящий трафик необходимо направить на edge router (порты 80, 443) при помощи правил DNAT на выделенный адрес во внутренней сети. Этот адрес поднимается при помощи MetalLB в L2-режиме на выделенных frontend-узлах.

Настройка правил DNAT/SNAT на edge gateway

1. Перейдите на вкладку «Networking» → «Edge Gateways», откройте edge gateway:

vmw VMware Cloud Director

Data Centers

Applications

Networking

Networks

Edge Gateways

Data Center Groups

Security Tags

	Name	Status	Scope
☐	EDGE-flant-df-lab_DC1	Normal	flant-df-lab_DC1

2. Перейдите на вкладку «Services» → «NAT»:

vmw VMware Cloud Director

Data Centers

Applications

Networking

Networks

Edge Gateways

Data Center Groups

Security Tags

All Edge Gateways > EDGE-flant-df-lab_DC1

EDGE-flant-df-lab_DC1

OPEN IN VDC CONTEXT

Configuration

General

Rate Limiting

Services

Firewall

NAT

IPSec VPN

NEW

	Name	Category	State
☐	DNAT_HTTP	User defined	Enabled
☐	DNAT_HTTPS	User defined	Enabled

3. Добавьте следующие правила:

NAT										
Rules Logs										
NEW										
	Name	Category	State	NAT Action	External IP	Internal IP	Application	External Port	Destination IP	
☐	NAT-frontend-http	User de...	Active	DNAT		192.168.199.10	HTTP	80	-	
☐	NAT-frontend-https	User de...	Active	DNAT		192.168.199.10	HTTPS	443	-	
☐	NAT-master-0	User de...	Active	DNAT		192.168.199.2	SSH	22	-	

Первые два правила используются для входящего трафика, а третье — для доступа по SSH к узлу с control plane (без этого правила установка будет невозможна).

4. Чтобы виртуальные машины могли выходить в интернет, настройте правила SNAT, следуя примеру:

NAT										
Rules Logs										
NEW										
	Name	Category	State	NAT Action	External IP	Internal IP	Application	External Port	Destination IP	Logging
☐	SNAT	User de...	Active	SNAT		192.168.199.0/24	-	Any	-	No

Данное правило позволит виртуальным машинам из подсети 192.168.199.0/24 выходить в интернет.

Выпуск сертификатов Let's Encrypt

i Инструкция актуальна только для схемы размещения **Standard**.

При использовании cert-manager и ACME-проверки типа HTTP-01 может потребоваться настройка Hairpin NAT (NAT loopback) на Edge Gateway.

Это необходимо в сценарии, когда доменное имя Ingress резолвится во внешний IP-адрес Edge Gateway, а запрос к этому адресу выполняется **изнутри** кластера (из сети узлов). В таком случае трафик должен быть корректно возвращён обратно во внутреннюю сеть на адрес Ingress (например, адрес MetalLB).

Это важно для выпуска сертификатов Let's Encrypt, так как cert-manager выполняет предварительную проверку доступности challenge URL до обращения к ACME-провайдеру. Если URL недоступен изнутри кластера, выпуск сертификата не начнётся, даже если снаружи адрес открывается корректно.

Пример (сеть кластера 192.168.199.0/24):

- внутренняя сеть узлов: 192.168.199.0/24;
- внешний IP-адрес Edge Gateway: 194.117.83.19;
- внутренний адрес Ingress (например, MetalLB): 192.168.199.251.

В этом случае необходимо настроить Hairpin NAT для трафика из сети 192.168.199.0/24, направленного на внешний IP 194.117.83.19, с трансляцией на внутренний адрес Ingress 192.168.199.251.

i Для Edge Gateway на базе NSX-V настройка Hairpin NAT может быть обязательной.

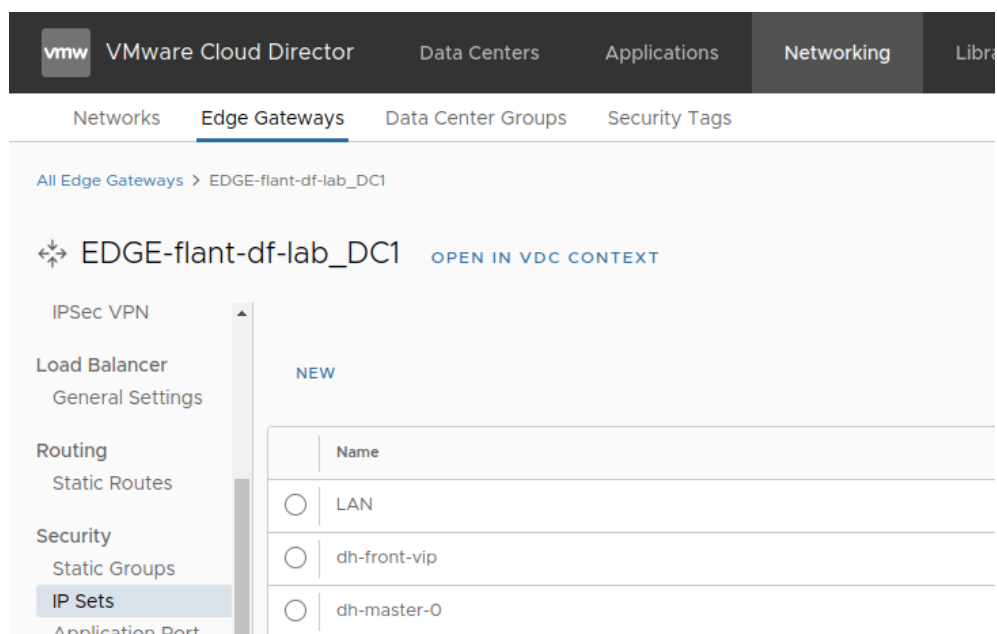
Для NSX-T loopback-сценарий часто поддерживается по умолчанию, но фактическое поведение зависит от конфигурации VMware Cloud Director.

Настройка firewall

i Инструкция актуальна только для схемы размещения **Standard**.

После настройки DNAT настройте firewall. Сначала необходимо настроить наборы IP-адресов (IP sets).

1. Перейдите на вкладку «Security» → «IP Sets»:



2. Создайте следующий набор IP (предполагается, что адрес MetalLB будет .10 а адрес узла с control plane — .2):

Edit IP Set

Name *

LAN

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR ⓘ

192.168.199.0/24

ADD

MODIFY

REMOVE

↶ UNDO

DISCARD

SAVE

Edit IP Set

Name *

dh-front-vip

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR

192.168.199.10

ADD

MODIFY

REMOVE

UNDO

DISCARD

SAVE

Edit IP Set

Name *

dh-master-0

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR

192.168.199.2

ADD

MODIFY

REMOVE

UNDO

DISCARD

SAVE

3. Добавьте следующие правила firewall:

Configuration

General

Rate Limiting

Services

Firewall

NAT

IPSec VPN

Load Balancer

General Settings

Routing

Static Routes

EDIT RULES

EDIT

DELETE

	#	Name	Category	State	Applications	Source	Destination	Action
☐	1	SNAT_LAN	User defined	Enabled	-	LAN	Any	Allow
☐	2	ACCESS_HTTP/HTTPS	User defined	Enabled	HTTPS, HTTP	Any	dh-front-vip	Allow
☐	3	ACCESS_SSH	User defined	Enabled	SSH	Any	dh-master-0	Allow
☐	4	Icmp	User defined	Enabled	ICMP ALL	Any	Any	Allow
☒		default_rule	Default	Enabled	-	Any	Any	Drop

Шаблон виртуальной машины

⚠ Работоспособность провайдера подтверждена только для шаблонов виртуальных машин на базе Ubuntu 22.04.

⚠ Отключите vApp/Guest OS Customization (Guest Customization, vApp Customization и аналогичные механизмы) для шаблона и виртуальных машин кластера в VMware Cloud Director. DKP выполняет первичную настройку узлов через `cloud-init` (datasource OVF/VMware GuestInfo). Включенная customization может конфликтовать с `cloud-init` и приводить к некорректной инициализации узла.

⚠ Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.

⚠ На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

В примере используется OVA-файл, предоставляемый Ubuntu, с двумя исправлениями. Исправления необходимы для корректного заказа CloudPermanent-узлов и для возможности подключать диски, созданные CSI.

Подготовка шаблона из OVA-файла

1. Скачайте OVA-файл:

	jammy-server-cloudimg-amd64.manifest	2024-02-21 13:01	18K	Package manifest file
	jammy-server-cloudimg-amd64.ova	2024-02-21 13:01	600M	VMware/Virtualbox OVA
	jammy-server-cloudimg-amd64.squashfs	2024-02-21 13:02	412M	Ubuntu Server 22.04 LTS (Jammy Jellyfish) dai

2. Перейдите на вкладку «Libraries» → «Catalogs» → «Каталог организации»:

VMware Cloud Director											
Data Centers Applications Networking Libraries Administration Monitor More											
All Catalogs > flant-df-lab_DC1											
flant-df-lab_DC1 SHARE PUBLISH SETTINGS SYNC DELETE ALL ACTIONS											
General											
vApp Templates											
Media & Other											
Metadata											
Name	Description	Status	Expired	Catalog Type	Catalog Publish Scope	Owner	Created On	Expires			
CentOS-Stream-9	https://cloud.centos.org/centos/9-stream/x86_64/...	Failed Creati...	No	Local	-	flant-df-lab_d...	11/30/2023, 07:32:27 PM	Never			
debian-11		Powered Off	No	Local	-	flant-df-lab_d...	02/21/2024, 02:11:44 PM	Never			
debian-11-cloud-template-2		Powered Off	No	Local	-	flant-df-lab_d...	02/21/2024, 09:19:19 AM	Never			
debian_ovas		Powered Off	No	Local	-	flant-df-lab_d...	02/07/2024, 10:34:51 AM	Never			
focal-server-cloudimg-amd64	https://cloud-images.ubuntu.com/focal/20231229/f...	Failed Creati...	No	Local	-	flant-df-lab_d...	11/30/2023, 05:13:58 PM	Never			
nick-password		Powered Off	No	Local	-	flant-df-lab_d...	12/06/2023, 05:22:42 PM	Never			

3. Выберите скачанный шаблон и загрузите его в каталог:

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Select Source

Enter a URL from which to upload directly an OVF file.

URL

Browse to a location accessible from your computer, such as a local hard drive, a network share or a CD/DVD drive and select an OVF/OVA and all related files.

Browse

File(s):

- jammy-server-cloudimg-amd64 (1).ova

CANCEL

NEXT

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Review Details

Verify the OVF template details.

Product	Ubuntu 22.04 Server (20240221)
Version	-
Vendor	-
Download size	599.65 MB
Size on disks	0 B (thin provisioned) 10 GB (thick provisioned)
Description	-

CANCEL

PREVIOUS

NEXT

Страница 315

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Select vApp Template Name

A vApp template is a virtual machine image that is loaded with an operating system, applications, and data. These templates ensure that virtual machines are consistently configured across an entire organization.

Name *

ubuntu-22.04-20240221

Description

Ubuntu 22.04 Server (20240221)
<https://cloud-images.ubuntu.com/jammy/20240221/jammy-server-cloudimg-amd64.ova>

Catalog *

flant-df-lab_DC1

CANCEL

PREVIOUS

NEXT

4. Создайте виртуальную машину из шаблона:

New VM

Name *

ubuntu-22.04-template

Computer Name *

ubuntu2204temp

Description

Type

☐ New

☒ From Template

Power on

☒

Templates

All VM templates

	Name	vApp Name	Catalog	OS	Compute	Storage
☐	ubuntu-jammy-22.04-cloudi...	ubuntu-jammy-22.04-cloudi...	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 4 Memory 8 GB	Policy
☒	ubuntu-jammy-22.04-cloudi...	ubuntu-22.04-20240221	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 2 Memory 1 GB	Policy
☐	ubuntu-jammy-22.04-cloudi...	Ubuntu 22.04 Server (2023...	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 2 Memory 1 GB	Policy

CANCEL

OK

Страница 316

New VM

Storage Policy

SSD-dc1-pub1-cl1 (VDC Default)

Compute

Sizing Policy

payg-4-8

Virtual CPUs

4

Cores per socket

1

Number of sockets

4

Memory

8 GB

NICs

Primary NIC	NIC	Connected	Network Adapter Type	Network	IP Mode	IP Address	MAC Address
+	0	☒	VMXNET3	LOCAL	DHCP		00:50:56:0c:17

CANCEL

OK

⚠ Укажите пароль по умолчанию и публичный ключ. Они понадобятся, чтобы войти в консоль виртуальной машины.

New VM

Custom Properties

A Unique Instance ID for this instance

id-ovf

Specifies the instance id. This is required and used to determine if the machine should take "first boot" actions

Uri to seed instance data from

This field is optional, but indicates that the instance should "seed" user-data and meta-data from the given uri. If set to 'http://tinyurl.com/sm-' is given, meta-data will be pulled from http://tinyurl.com/sm-meta-data and user-data from http://tinyurl.com/sm-user-data. Leave this empty if you do not want to seed from a uri.

ssh public keys

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDTXjTmx3hq2EPDQHWSJN7By1VNFZ8coll5tEeZDBVYae9Oxq4FZsKCb1e

This field is optional, but indicates that the instance should populate the default user's 'authorized_keys' with this value

Encoded user-data

In order to fit into a xml attribute, this value is base64 encoded . It will be decoded, and then processed normally as user-data.

Default User's password

123456

If set, the default user's password will be set to this value to allow password based login. The password will be good for only a single login. If set to the string 'RANDOM' then a random password will be generated, and written to the console.

End User License Agreements

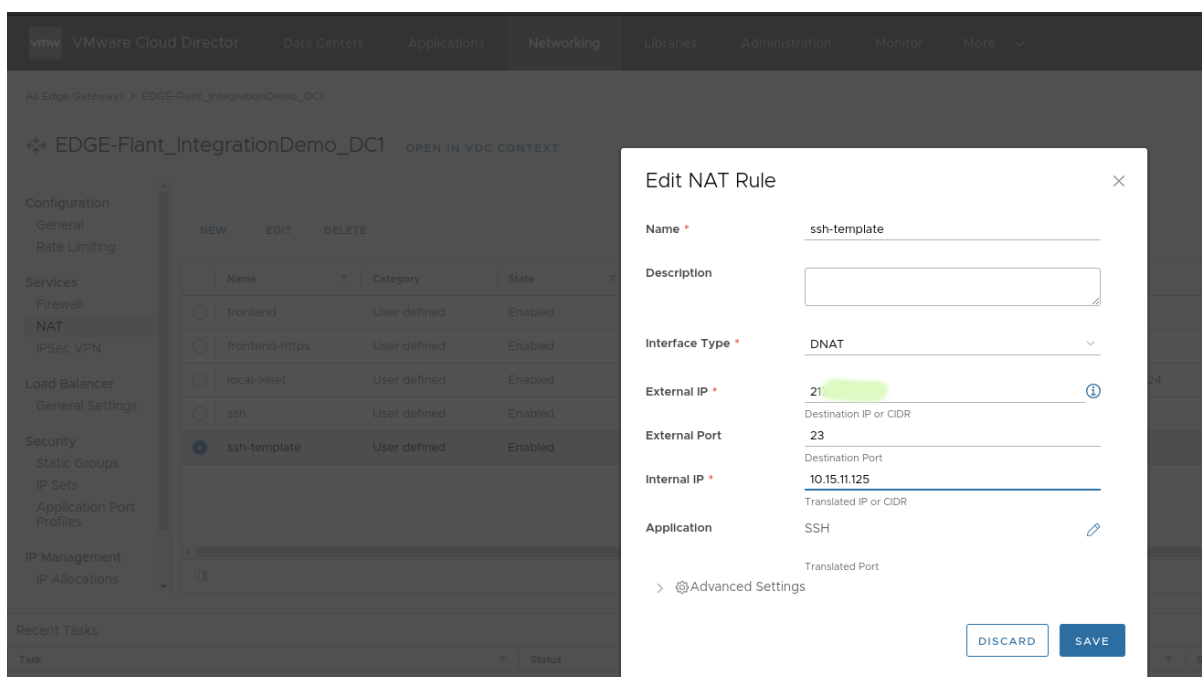
There are no EULAs to review.

CANCEL

OK

Для подключения к виртуальной машине выполните следующие шаги:

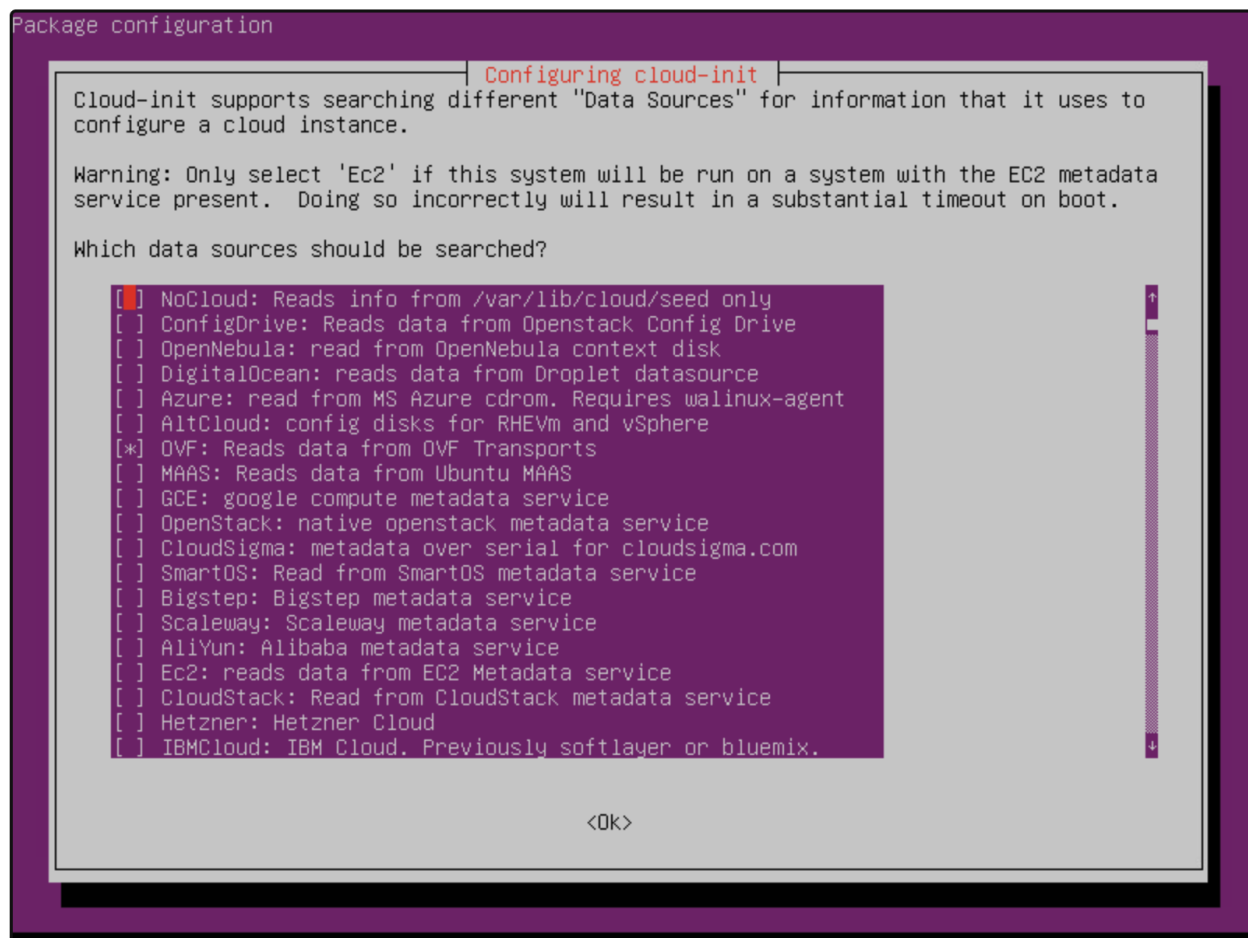
- 1. Запустите виртуальную машину.
- 2. Дождитесь получение IP-адреса.
- 3. Пробросьте порт 22 до виртуальной машины:



Войдите на виртуальную машину по SSH и выполните следующие команды:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91-vmware_cust.cfg
dpkg-reconfigure cloud-init
```

В появившемся диалоговом окне оставьте галочку только у **OVF: Reads data from OVF transports**. Остальные пункты отключите:



Убедитесь, что в конфигурации **cloud-init** задан параметр **datasource_list**, с помощью следующей команды:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

Если вывод окажется пустым, выполните следующую команду:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

Выполните оставшиеся команды:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

Настройка шаблона в VCD

- 1. Выключите виртуальную машину и удалите все заполненные поля «Guest Properties»:

ubuntu-22.04
Powered off

POWER ONPOWER OFFLAUNCH WEB CONSOLELAUNCH REMOTE CONSOLEALL ACTIONS

GeneralSecurity TagsHardwareRemovable MediaHard DisksComputeNICsGuest OSCustomizationGuest PropertiesMetadataMonitorTasksEvents

EDIT

A Unique Instance ID for this instance	Specifies the instance id. This is required and used to determine if the machine should take "first boot" actions
-	ubuntuguest Specifies the hostname for the appliance
Url to seed instance data from	This field is optional, but indicates that the instance should 'seed' user-data and meta-data from the given url. If set to 'http://tinyurl.com/sm-' is given, meta-data will be pulled from http://tinyurl.com/sm-meta-data and user-data from http://tinyurl.com/sm-user-data. Leave this empty if you do not want to seed from a url.
ssh public keys	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDTXJTmx3hq2EPDQHWSJN7By1VNFZ8coll5tEeZD BVYAe9Oxq4FZsKCb1aGlSkDaiAHTxrbd2efoJTcPQLBSBM79dcELtqfKj9dtjy4S1W0mydvWb2o WLnvoaZX/H6pqjz8jrJAKXwXj2pWCOzXerwk9oSi4fCE7VbqsfT4bBfv27FN4/Vqa6iWiCc71oJo pL9DIdtUlYDVUgOZOa+12J4hPCCSqEJK/r+ToHQbOWxbC5/OAufXDw2W1vkVeaZUur5xwwAx lb3wM3WoS3BbwNIDYg9UB2D8+EZgNz1CCCPsy1ELIn7q8RnrTp0+H8V9LoWHSgh3VCWeW8 C/MnTW90IR This field is optional, but indicates that the instance should populate the default user's 'authorized_keys' with this value
Encoded user-data	In order to fit into a xml attribute, this value is base64 encoded . It will be decoded, and then processed normally as user-data

Edit Guest Properties for ubuntu-22.04

A Unique Instance ID for this instance

Url to seed instance data from

ssh public keys

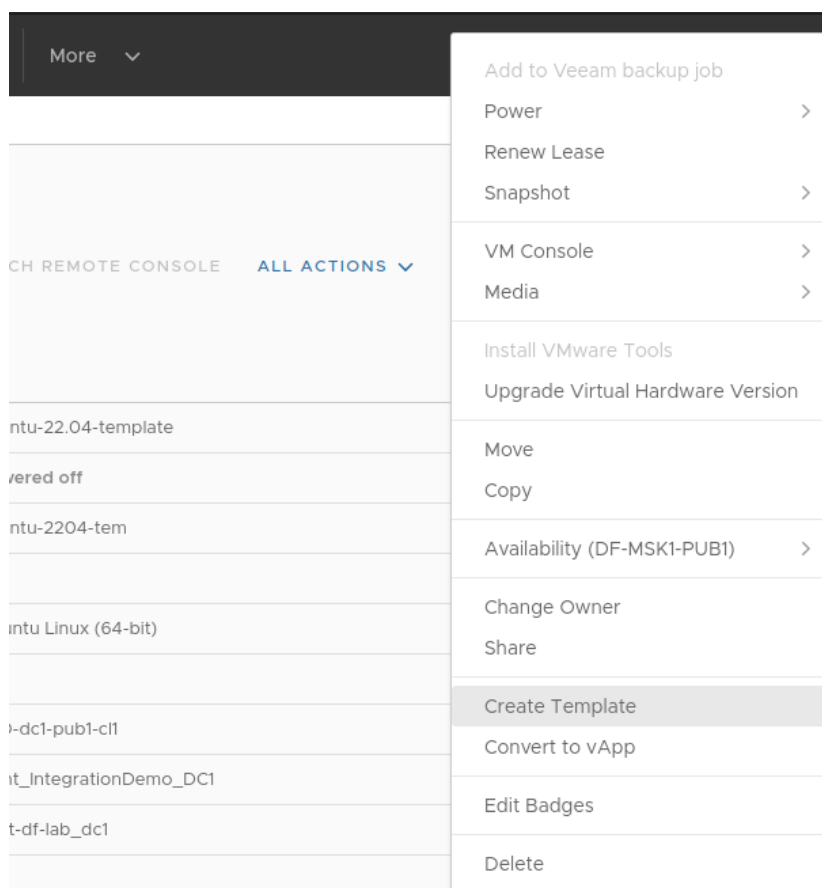
Encoded user-data

Default User's password

CANCEL

OK

- 2. Создайте шаблон виртуальной машины:



Add to Catalog: ubuntu-22.04-template

Add this vApp to catalog:

Catalog: * flant-df-lab_DC1

ⓘ This catalog is local to your organization.

Overwrite catalog item: ☐

Name * ubuntu-22.04-template

Description

When using this template:

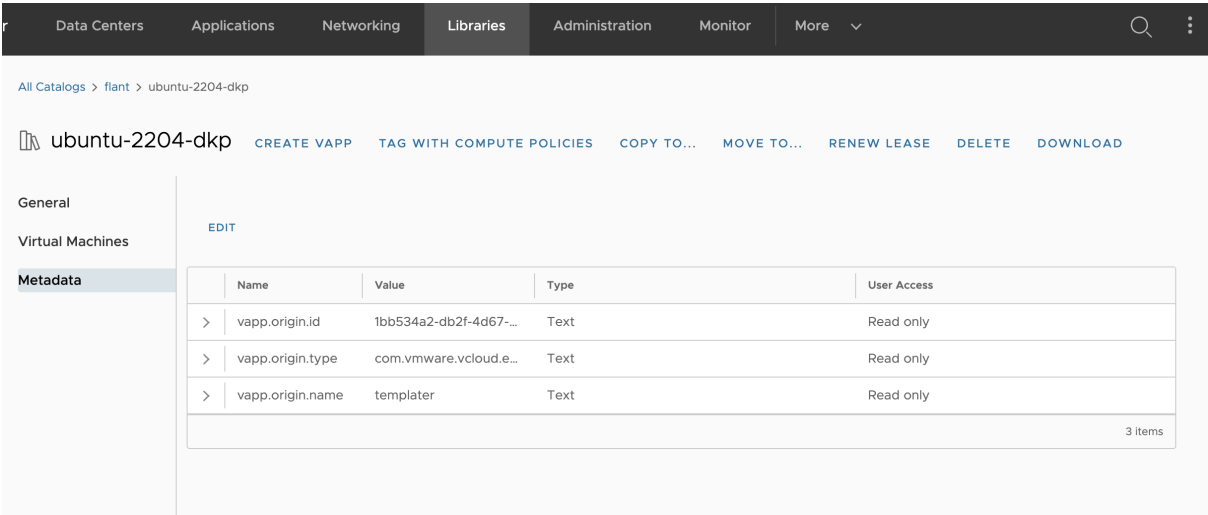
☐ Make identical copy
 ☒ Customize VM settings

This setting applies when creating a vApp based on this template. It is ignored when building a vApp using individual VMs from this template.

CANCEL
OK

3. В созданном шаблоне перейдите на вкладку «Metadata» и добавьте шесть полей:

- `guestinfo.metadata;`
- `guestinfo.metadata.encoding;`
- `guestinfo.userdata;`
- `guestinfo.userdata.encoding;`
- `disk.enableUUID;`
- `guestinfo.hostname.`

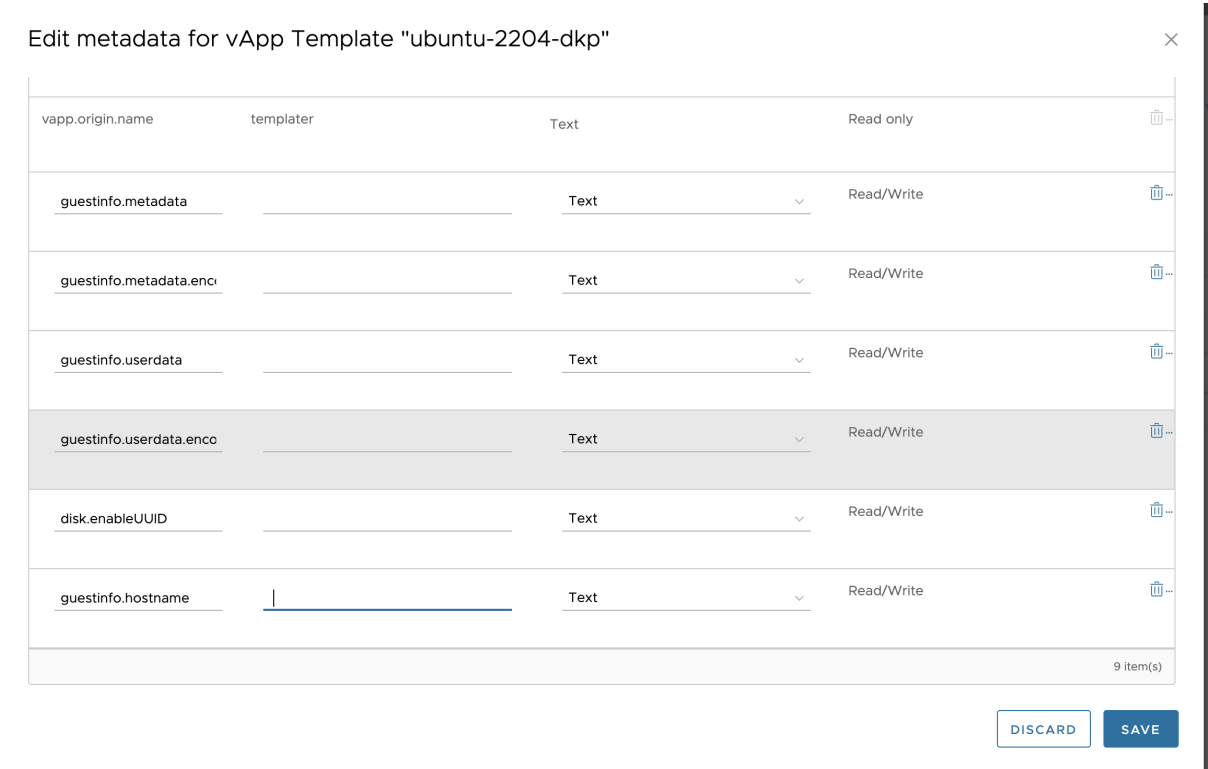


Для **каждого** поля в форме добавления/редактирования укажите:

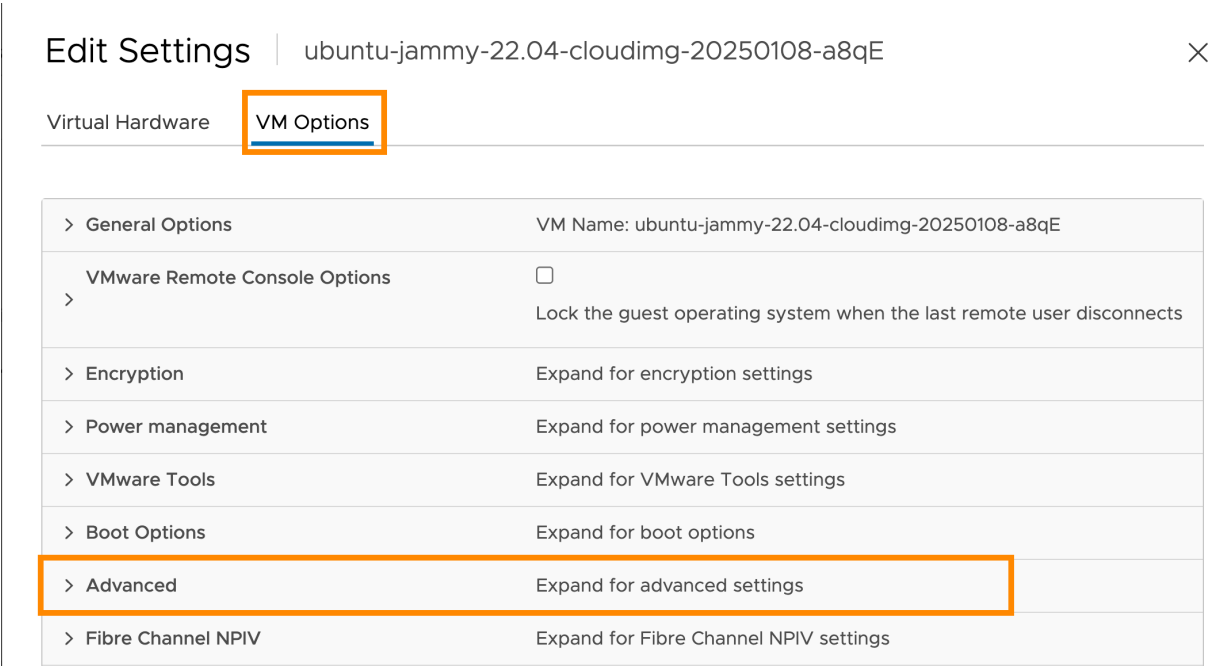
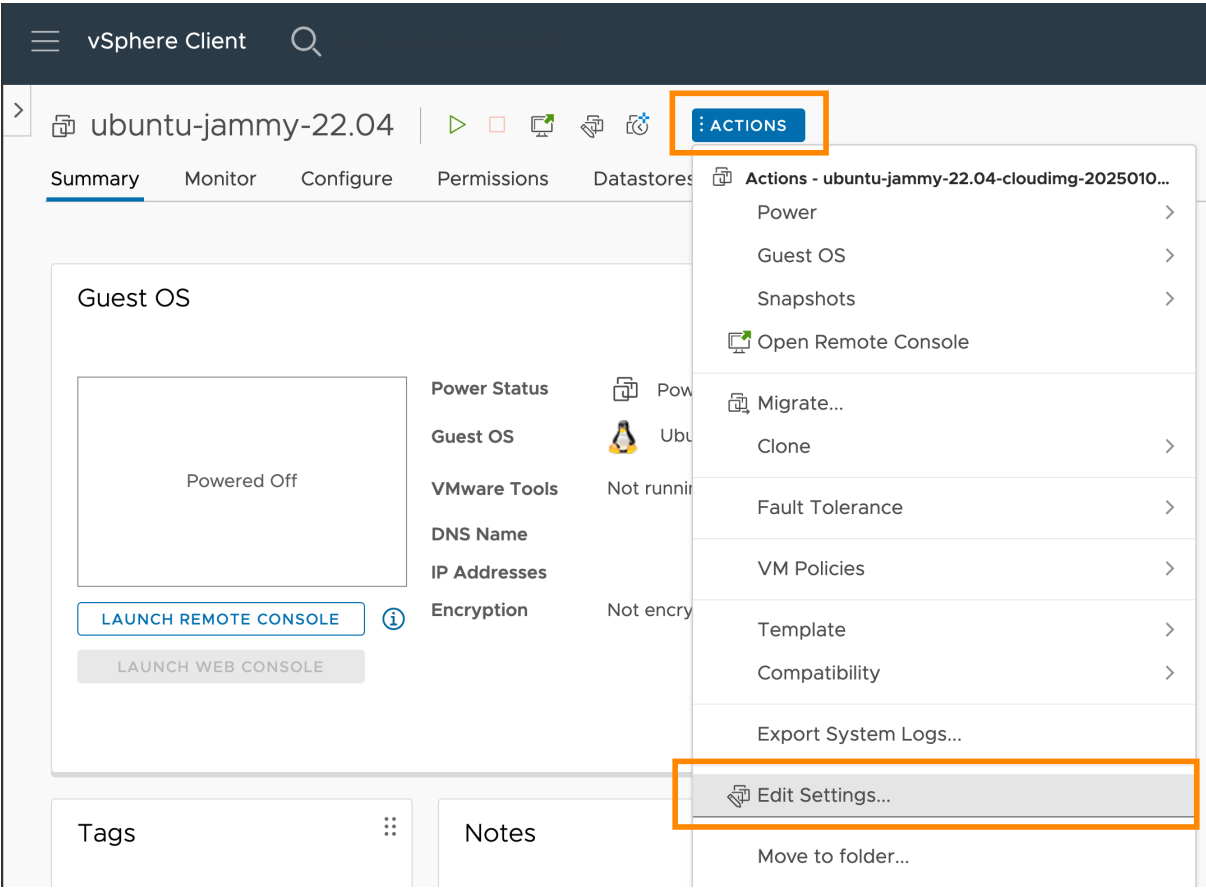
- «Type»: Text (текстовое значение);
- «User access»: Read/Write;
- «Value»: один пробел (space).

i

Интерфейс VCD может не сохранять метаданные с пустым значением. Пробел используется как техническое заполнение и не влияет на работу. Фактические значения будут подставлены автоматически при создании виртуальных машин.



4. В панели управления vCenter для шаблона включите параметр `disk.EnableUUID`:



Edit Settings | ubuntu-jammy-22.04-cloudimg-20250108-a8qE

Virtual Hardware | VM Options

> General Options

VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE

VMware Remote Console Options

☐

> Lock the guest operating system when the last remote user disconnects

> Encryption

Expand for encryption settings

> Power management

Expand for power management settings

> VMware Tools

Expand for VMware Tools settings

> Boot Options

Expand for boot options

< Advanced

Settings

☐ Disable acceleration

☒ Enable logging

Debugging and statistics

Run normally

>

Swap file location

☒ Default

Use the settings of the cluster or host containing the virtual machine.

☐ Virtual machine directory

Store the swap files in the same directory as the virtual machine.

☐ Datastore specified by host

Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.

Configuration Parameters

EDIT CONFIGURATION...

Configuration Parameters

⚠ Modify or add configuration parameters as needed for experimental features or as instructed by technical support. Empty values will be removed (supported on ESXi 6.0 and later).

ADD CONFIGURATION PARAMS

Name	Value
nvram	ubuntu-jammy-22.04-clc
svga.present	TRUE
pciBridge0.present	TRUE
pciBridge1.present	TRUE

Configuration Parameters



Modify or add configuration parameters as needed for experimental features or as instructed by technical support. Empty values will be removed (supported on ESXi 6.0 and later).

ADD CONFIGURATION PARAMS

Add New Configuration Params

Name	Value
disk.EnableUUID	TRUE

Name	Value
nvram	ubuntu-jammy-22.04-clc
svga.present	TRUE
pciBridge0.present	TRUE
pciBridge4.present	TRUE
pciBridge4.virtualDev	pcieRootPort
pciBridge4.functions	8
pciBridge5.present	TRUE
pciBridge5.virtualDev	pcieRootPort
pciBridge5.functions	8
pciBridge6.present	TRUE
pciBridge6.virtualDev	pcieRootPort
pciBridge6.functions	8
pciBridge7.present	TRUE
pciBridge7.virtualDev	pcieRootPort
pciBridge7.functions	8

CANCEL

OK

Использование хранилища

- VCD поддерживает CSI. Диски создаются как VCD Independent Disks.
- Guest property `disk.EnableUUID` должно быть разрешено для используемых шаблонов виртуальных машин.
- Deckhouse Kubernetes Platform поддерживает изменение размера дисков с версии v1.59.1.

Использование балансировщика нагрузки

- Компоненты DKP поддерживают ресурсов Service типа LoadBalancer при установке в VMware Cloud Director (VCD).
- В качестве балансировщика используется VMware NSX Advanced Load Balancer (ALB или Avi Networks).
- Поддержка доступна **только** при использовании платформы виртуализации сети NSX-T.
- Балансировка должна быть включена на Edge Gateway вашим провайдером VCD. Проверить, включена ли балансировка, можно в разделе Edge Gateway → Load Balancer → General Settings — параметр `State` должен быть в статусе `Active`.
- Если балансировщик был активирован после успешного создания кластера DKP, компоненты автоматически подхватят изменения в течение часа (дополнительных действий не требуется).
- Для каждого открытого порта создаётся связь Pool + Virtual Service.
- При наличии межсетевого экрана необходимо создать разрешающее правило для внешнего IP-адреса балансировщика и соответствующих портов.

Дополнительные ресурсы

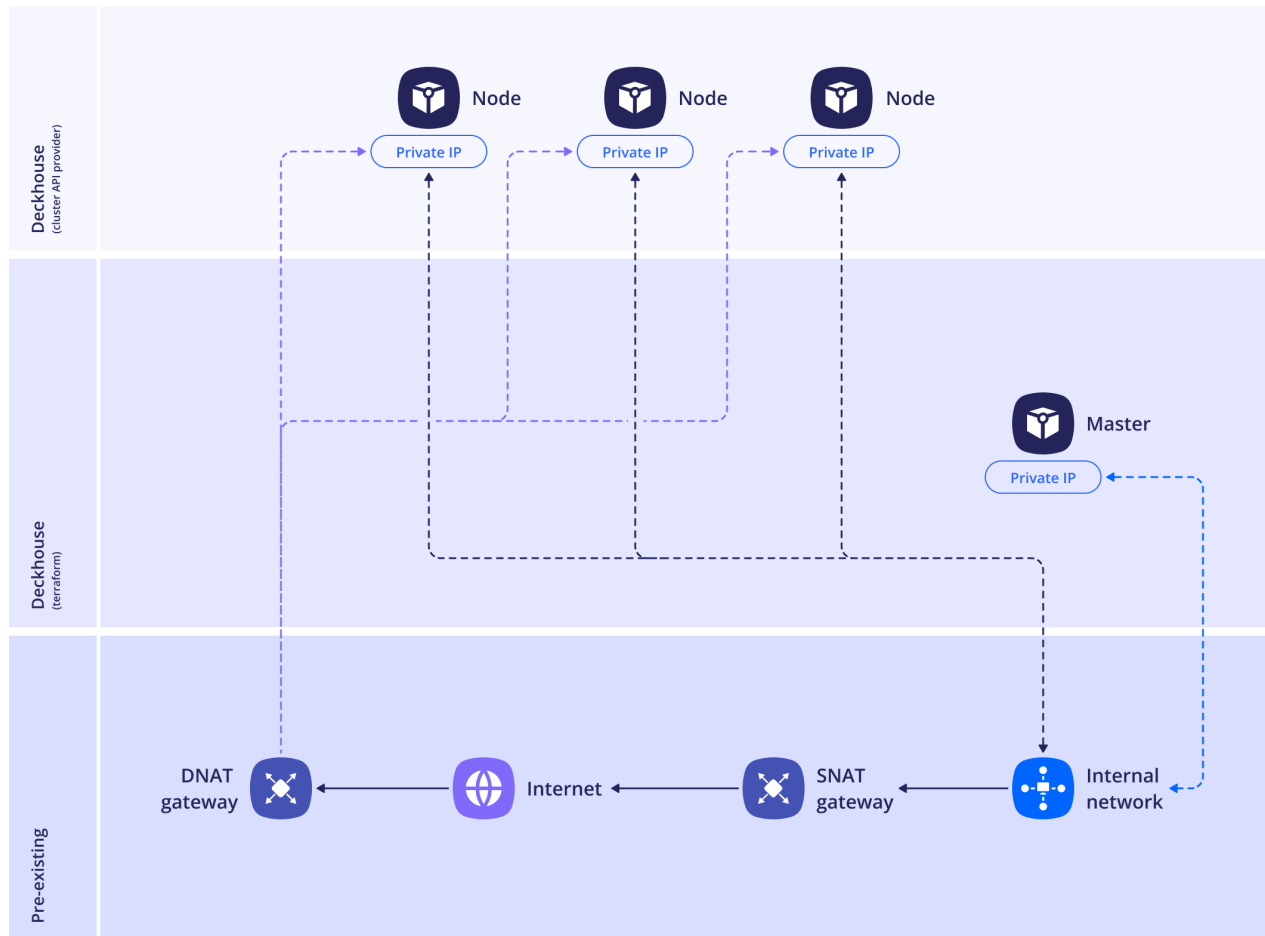
- Документация модуля `cert-manager`
- Документация модуля `metalb`

Схемы размещения и настройка VMware Cloud Director

Схемы размещения

Deckhouse Kubernetes Platform поддерживает две схемы размещения ресурсов в VCD.

Standard



Пример конфигурации схемы размещения:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
instanceClass:
  storageProfile: "Fast vHDD"
  sizingPolicy: 4cpu8mem
  template: "catalog/Ubuntu 22.04 Server"
  mainNetwork: internal
  mainNetworkIPAddresses:
    - 192.168.199.2
  mainNetwork: internal
```

WithNAT



При использовании данной схемы размещения необходимо уточнить у администратора тип платформы сетевой виртуализации и указать его в параметре `edgeGateway.type`. Поддерживаются два варианта: `NSX-T` и `NSX-V`.

Для обеспечения административного доступа к узлам кластера разворачивается бастийон. Параметры для его настройки описываются в секции `bastion`.

Если Edge Gateway работает на базе `NSX-T`, в созданной сети для узлов автоматически активируется DHCP-сервер. Он будет выделять IP-адреса, начиная с 30-го адреса в подсети и до предпоследнего (перед broadcast-адресом). Начальный адрес DHCP-пула можно изменить с помощью параметра `internalNetworkDHCPPoolStartAddress`.

Если используется `NSX-V`, DHCP необходимо настроить вручную. В противном случае узлы, ожидающие получение IP-адреса по DHCP, не смогут его получить.



Не рекомендуется использовать динамическую адресацию для первого master-узла совместно с `NSX-V`.

Схема размещения предполагает автоматическое создание следующих правил NAT:

- SNAT — трансляция адресов внутренней сети узлов во внешний адрес, указанный в параметре `edgeGateway.externalIP`.
- DNAT — трансляция внешнего адреса и порта, заданных в параметрах `edgeGateway.externalIP` и `edgeGateway.externalPort`, на внутренний IP-адрес бастийона по порту 22 (протокол TCP) для обеспечения административного доступа по SSH.



Если Edge Gateway обеспечивается средствами `NSX-V`, то для построения правил необходимо указать имя и тип сети, к которым правило будет привязано в свойствах `edgeGateway.NSX-V.externalNetworkName` и `edgeGateway.NSX-V.externalNetworkType` соответственно. Как правило, это сеть, подключённая к Edge Gateway в разделе `Gateway Interfaces` и имеющая внешний IP-адрес.

Дополнительно возможно создание правил брандмауэра отдельным свойством `createDefaultFirewallRules`.



Если Edge Gateway обеспечивается средствами `NSX-T`, то существующие в Edge Gateway правила будут перезаписаны. Предполагается, что использование данной опции подразумевает размещение одного кластера на Edge Gateway.

Будут созданы следующие правила:

- Разрешение любого исходящего трафика;
- Разрешение входящего трафика по протоколу TCP и 22 порту для соединения с узлами кластера по SSH;
- Разрешение любого входящего трафика по протоколу ICMP;
- Разрешение входящего трафика по протоколам TCP и UDP и портам 30000-32767 для использования `NodePort`.

Пример конфигурации схемы размещения с использованием **NSX-T**:

```
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
      - 192.168.199.2
```

Пример конфигурации схемы размещения с использованием **NSX-V**:

```

apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
  mainNetworkIPAddresses:
    - 192.168.199.2

```

Конфигурация

Интеграция осуществляется с помощью ресурса VCDClusterConfiguration, который описывает конфигурацию облачного кластера в VCD и используется системой виртуализации, если управляющий слой (control plane) кластера размещён в системе. Отвечающий за интеграцию модуль DKP настраивается автоматически, исходя из выбранной схемы размещения.

Чтобы изменить конфигурацию в запущенном кластере, выполните следующую команду:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

Пример конфигурации:

```

apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    template: Org/Templates/ubuntu-focal-20.04
    sizingPolicy: 16cpu32ram
    storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true

```

Количество и параметры процесса заказа машин в облаке настраиваются в кастомном ресурсе `NodeGroup`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference`). Инстанс-класс для cloud-провайдера VCD — это кастомный ресурс `VCDInstanceClass`, в котором указываются конкретные параметры самих машин.

Ниже представлен пример конфигурации `VCDInstanceClass` для эфемерных узлов cloud-провайдера VMware Cloud Director.

Пример конфигурации кастомного ресурса `VCDInstanceClass`

```

apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova

```

Storage

Для каждого `Datastore` и `DatastoreCluster` из зон (зоны) автоматически создаётся `StorageClass`.

Имя `StorageClass`'а, который будет использоваться в кластере по умолчанию, можно настроить (параметр `default`) и отфильтровать ненужные `StorageClass`'ы (параметр `exclude`).

CSI

Подсистема хранения по умолчанию использует CNS-диски с возможностью изменения их размера на лету. Но также поддерживается работа и в legacy-режиме с использованием FCD-дисков. Поведение подсистемы устанавливается с помощью параметра `compatibilityFlag`.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-vcd`
- Кастомные ресурсы модуля `cloud-provider-vcd`
- Кастомные ресурсы модуля `node-manager`

Подключение и авторизация в VMware vSphere

Требования

Для корректной работы Deckhouse Kubernetes Platform с VMware vSphere необходимы:

- Доступ к vCenter;
- Пользователь с необходимым набором привилегий;
- Созданные теги и категории тегов в vSphere;
- Сети с DHCP и доступом в Интернет;
- Доступные shared Datastore на всех используемых ESXi;
- Версия vSphere — 7.x или 8.x с поддержкой механизма `Online volume expansion`;
- vCenter, доступный изнутри кластера с master-узлов;
- Созданный Datacenter, в котором должны быть настроены следующие объекты:
 1. VirtualMachine template:
 - Образ виртуальной машины должен использовать `Virtual machines with hardware version 15 or later` — это необходимо для работы `online resize`.
 - В образе должны быть установлены пакеты `open-vm-tools`, `cloud-init` и `cloud-init-vmware-guestinfo` — при использовании версии `cloud-init` ниже 21.3.
 2. Network:
 - Сеть должна быть доступна на всех ESXi, на которых планируется создание виртуальных машин.
 3. Datastore (один или несколько):
 - Datastore должен быть подключен ко всем ESXi, на которых планируется создание виртуальных машин.
 - На Datastore должен быть назначен тег из категории, указанной в параметре `zoneTagCategory` (по умолчанию — `k8s-zone`). Этот тег определяет зону.
 - Все Cluster в пределах одной зоны должны иметь доступ ко всем Datastore с той же зоной.
 4. Cluster:
 - В Cluster должны быть добавлены все используемые ESXi.
 - На Cluster должен быть назначен тег из категории, указанной в параметре `zoneTagCategory` (по умолчанию — `k8s-zone`). Этот тег определяет зону.
 5. Folder для создаваемых виртуальных машин:
 - Параметр опционален.
 - По умолчанию используется корневой каталог виртуальных машин.
 6. Role:
 - Роль должна содержать необходимый набор привилегий.
 7. User:
 - Пользователю должна быть назначена роль, указанная в предыдущем пункте.
- На созданный Datacenter должен быть назначен тег из категории, указанной в параметре `regionTagCategory` (по умолчанию — `k8s-region`). Этот тег определяет регион.

Требования к образу виртуальной машины

Для создания шаблона виртуальной машины (Template) рекомендуется использовать готовый cloud-образ/OVA-файл, предоставляемый вендором ОС:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (секция *Generic Cloud / OpenStack*)



Если вы планируете использовать дистрибутив отечественной ОС, обратитесь к вендору ОС для получения образа/OVA-файла.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Подготовка образа виртуальной машины

DKP использует `cloud-init` для настройки виртуальной машины после запуска.



Отключите VMware Guest OS Customization (а также любые механизмы vApp/OS customization, если они применимы в вашей схеме) для шаблона и виртуальных машин кластера в vSphere. DKP выполняет первичную настройку узлов через `cloud-init` (datasource VMware GuestInfo). Включенная customization может конфликтовать с `cloud-init` и приводить к некорректной инициализации узла.

Чтобы подготовить `cloud-init` и образ VM, выполните следующие действия:

1. Установите необходимые пакеты:

Если используется версия `cloud-init` ниже 21.3 (требуется поддержка VMware GuestInfo):

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
```

Если используется версия `cloud-init` 21.3 и выше:

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init
```

2. Проверьте, что в файле `/etc/cloud/cloud.cfg` установлен параметр `disable_vmware_customization: false`.
3. Убедитесь, что в файле `/etc/cloud/cloud.cfg` указан параметр `default_user`. Он необходим для

добавления SSH-ключа при запуске ВМ.

- Добавьте datasource VMware GuestInfo — создайте файл `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

```
datasource:
  VMware:
    vmware_cust_file_max_wait: 10
```

- Перед созданием шаблона ВМ сбросьте идентификаторы и состояние `cloud-init`, используя следующие команды:

```
truncate -s 0 /etc/machine-id &&
rm /var/lib/dbus/machine-id &&
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

- Очистите логи событий `cloud-init`:

```
cloud-init clean --logs --seed
```



После запуска виртуальной машины в ней должны быть запущены следующие службы, связанные с пакетами, установленными при подготовке `cloud-init`:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

Чтобы убедиться в том, что службы включены, используйте команду:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Пример ответа для включенных служб:

```
enabled
enabled
enabled
```



DKP создаёт диски виртуальных машин с типом `eagerZeroedThick`, но тип дисков созданных ВМ будет изменён без уведомления, согласно настроенным в vSphere `VM Storage Policy`. Подробнее можно прочитать в документации.



DKP использует интерфейс `ens192`, как интерфейс по умолчанию для виртуальных машин в vSphere. Поэтому, при использовании статических IP-адресов в `mainNetwork`, вы должны в образе ОС создать интерфейс с именем `ens192`, как интерфейс по умолчанию.

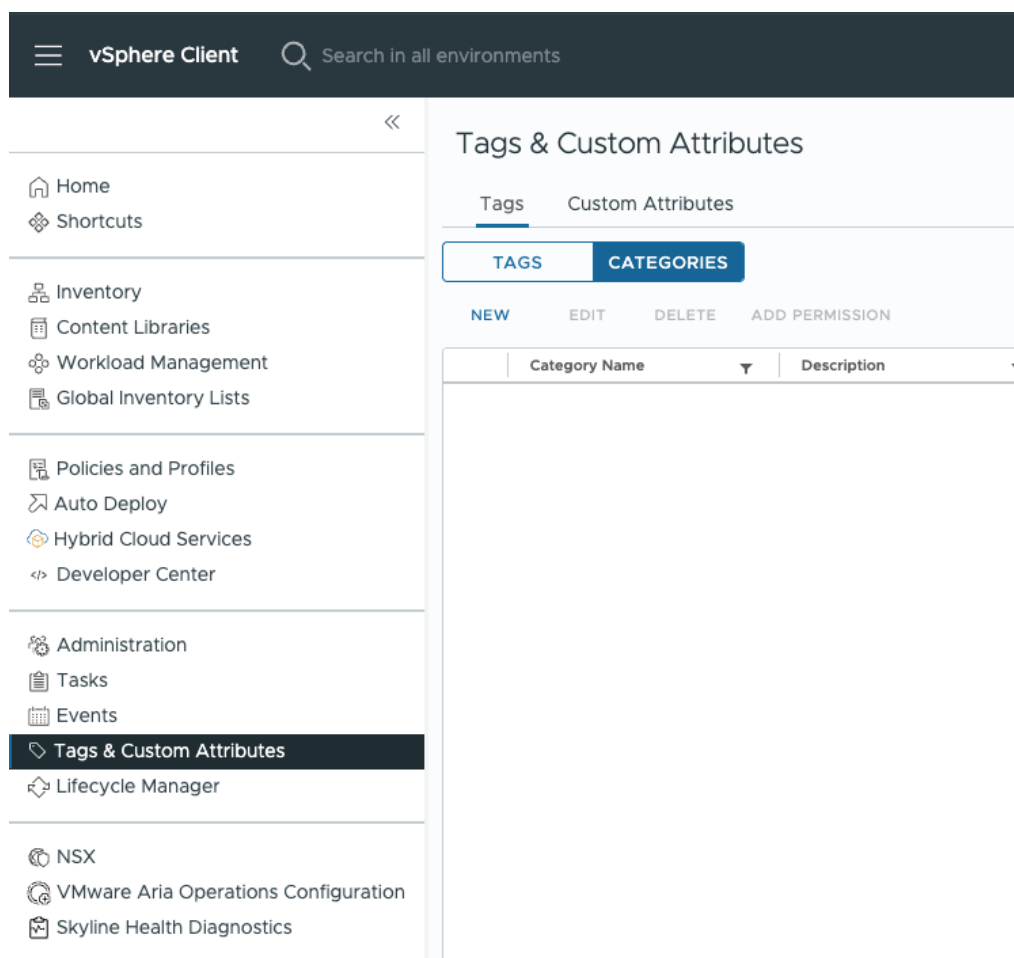
Конфигурация vSphere

Настройка через vSphere Client

Создание тегов и категорий тегов с использованием vSphere Client

В VMware vSphere нет понятий «регион» и «зона». «Регионом» в vSphere является Datacenter, а «зоной» — Cluster. Для создания этой связи используются теги.

- Откройте vSphere Client и перейдите в «Menu» → «Tags & Custom Attributes» → «Tags».



- Откройте вкладку «Categories» и нажмите «NEW». Создайте категорию для регионов (например `k8s-region`): установите значение «One tag» для параметра «Tags Per Object» и задайте связываемые типы, включая Datacenter.

- Создайте вторую категорию для зон (например, `k8s-zone`) с типами объектов Host, Cluster и Datastore.

Create Category

Category Name:

Description:

Tags Per Object:

☒ One tag
☐ Many tags

Associable Object Types:

☐ All objects
☐ Folder
☐ Datacenter
☐ Datastore Cluster
☐ Distributed Switch
☐ Content Library
☐ Network
☐ vApp
☒ Cluster
☒ Datastore
☐ Distributed Port Group
☒ Host
☐ Library Item
☐ Resource Pool
☐ Virtual Machine

CANCEL

CREATE

4. Перейдите на вкладку «Tags» и создайте минимум по одному тегу в категории региона и в категории зон (например, `test-region`, `test-zone-1`).

vSphere Client

Search in all environments

Home
Shortcuts
Inventory
Content Libraries
Workload Management
Global Inventory Lists
Policies and Profiles
Auto Deploy
Hybrid Cloud Services
Developer Center
Administration
Tasks
Events
Tags & Custom Attributes
Lifecycle Manager
NSX
VMware Aria Operations Configuration
Skyline Health Diagnostics

Tags & Custom Attributes

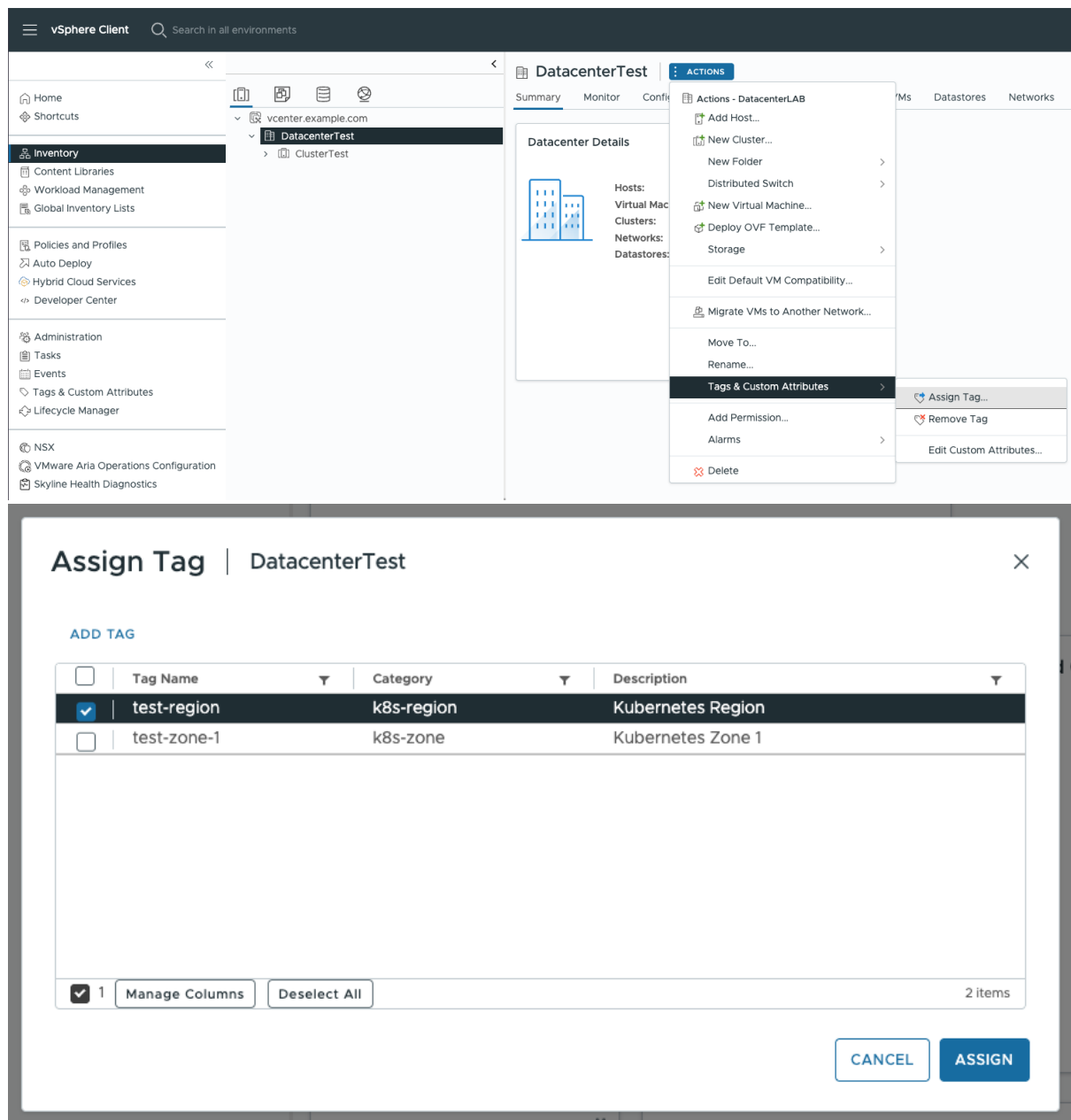
Tags
Custom Attributes

TAGS
CATEGORIES

NEW
EDIT
DELETE
ADD PERMISSION

	Tag Name	Category
☐	test-region	k8s-region
☐	test-zone-1	k8s-zone

5. Во вкладке «Inventory» выберите целевой Datacenter, перейдите в панель «Summary», откройте «Actions» → «Tags & Custom Attributes» → «Assign Tag» и назначьте тег региона. Повторите для каждого Cluster, на котором будут узлы, назначая соответствующие теги зон.

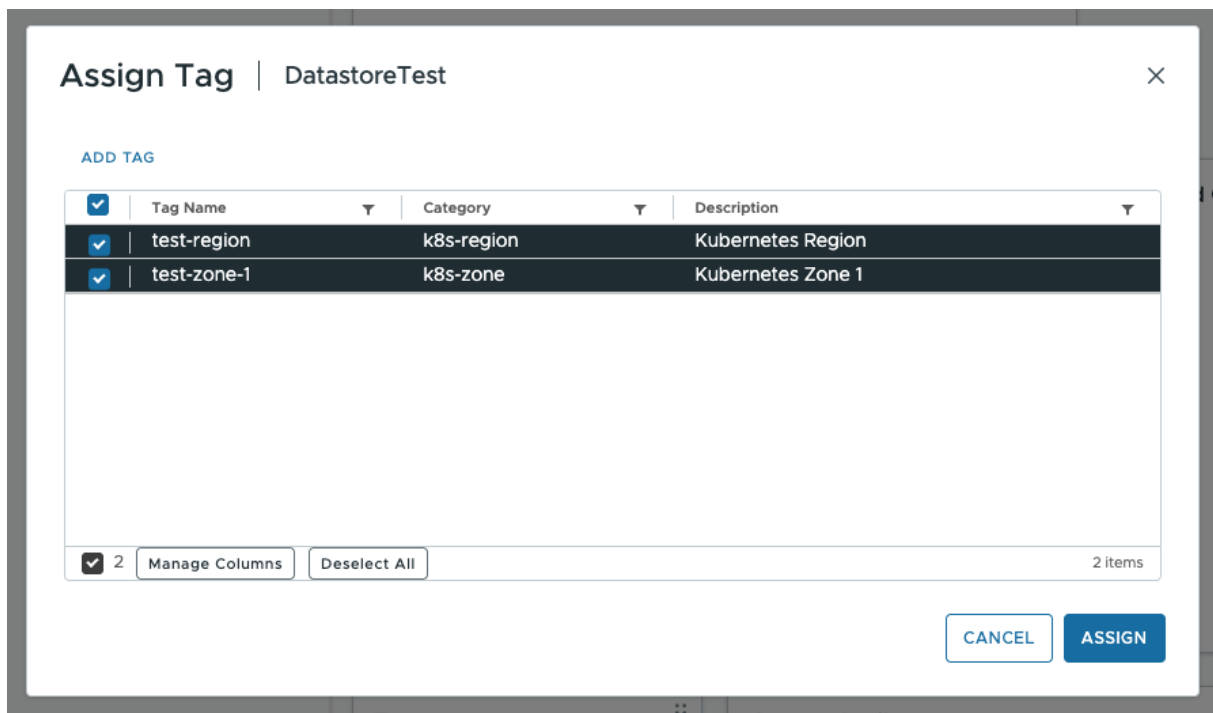


Настройка Datastore с использованием vSphere Client



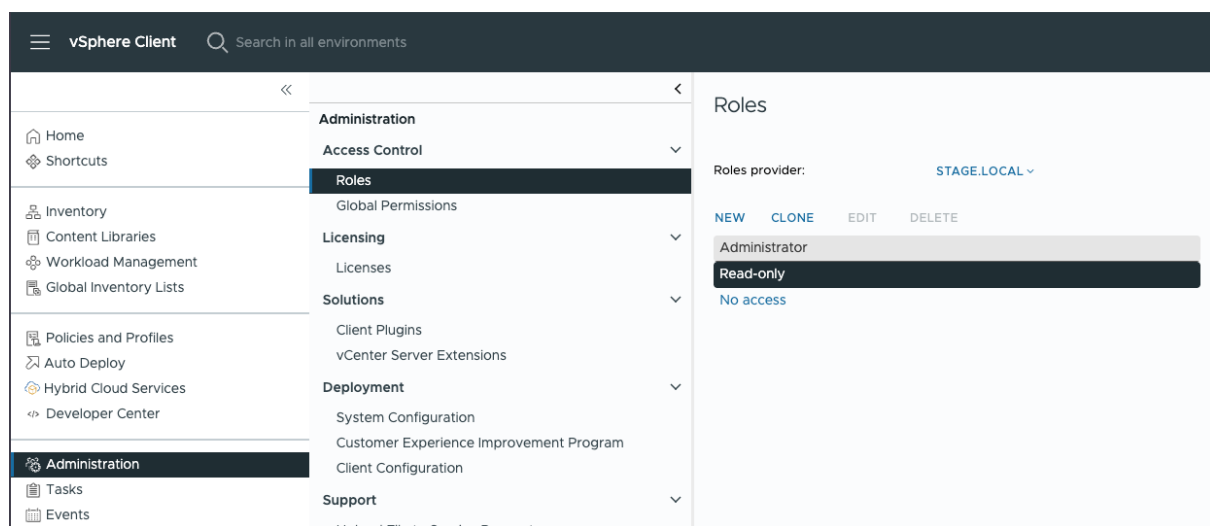
Для динамического заказа PersistentVolume необходимо, чтобы Datastore был доступен на **каждом** хосте ESXi в зоне (shared datastore).

Во вкладке «Inventory» выберите Datastore, перейдите в панель «Summary», затем откройте меню «Actions» → «Tags & Custom Attributes» → «Assign Tag». Назначьте Datastore тот же тег региона, что и у соответствующего Datacenter, а также тот же тег зоны, что и у соответствующего Cluster.



Создание и назначение роли с использованием vSphere Client

1. Перейдите в «Menu» → «Administration» → «Access Control» → «Roles».



2. Нажмите «NEW», введите имя роли (например, `deckhouse`) и добавьте привилегии из списка.

3. Назначьте роль для учётной записи Deckhouse, во вкладке «Menu» → «Administration» → «Access Control» → «Global Permissions» нажмите «ADD» и выберите пользователя и роль `deckhouse`.

Настройка через govc

Установка govc

Для дальнейшей настройки vSphere потребуется CLI-утилита govc.

После установки задайте переменные окружения для работы с vCenter.



Обязательно указывайте имя пользователя вместе с доменом, например: `username@vsphere.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Создание тегов и категорий тегов с использованием govc

В VMware vSphere нет понятий «регион» и «зона». «Регионом» в vSphere является Datacenter, а «зоной» — Cluster.

Для создания этой связи используются теги.

Создайте категории тегов с помощью команд:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Создайте теги в каждой категории. Если вы планируете использовать несколько «зон» (Cluster), создайте тег для каждой из них:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Назначьте тег «региона» на Datacenter:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>
```

Назначьте теги «зон» на объекты Cluster:

```
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
```

Настройка Datastore с использованием govc



Для динамического заказа PersistentVolume необходимо, чтобы Datastore был доступен на **каждом** хосте ESXi (shared datastore).

Для автоматического создания StorageClass в кластере Kubernetes назначьте созданные ранее теги «региона» и «зоны» на объекты Datastore:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Создание и назначение роли с использованием govc



Ввиду разнообразия подключаемых к vSphere SSO-провайдеров шаги по созданию пользователя в данной статье не рассматриваются.

Роль, которую предлагается создать далее, включает в себя привилегии из раздела «Список необходимых привилегий». При необходимости более гранулярных прав обратитесь в техподдержку Deckhouse.

Создайте роль с необходимыми привилегиями:

```
govc role.create deckhouse \
  Cns.Searchable \
  Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
  Folder.Create Folder.Delete Folder.Move Folder.Rename \
  Global.GlobalTag Global.SystemTag \
  InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
  InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
  InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
  InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
  InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
  Network.Assign \
  Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
  StorageProfile.View \
  System.Anonymous System.Read System.View \
  VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
  VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
  VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
  VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
  VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
  VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
  VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
  VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
  VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
  VirtualMachine.Config.UpgradeVirtualHardware \
  VirtualMachine.GuestOperations.Query \
  VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
  VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
  VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
  VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
  VirtualMachine.Inventory.Move \
  VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
  VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs \
  VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Назначьте пользователю роль на объекте vCenter.

 Обязательно указывайте имя пользователя вместе с доменом, например: `username@vsphere.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

 Для более детальной настройки прав обратитесь к официальной документации.

Дополнительные ресурсы

- Документация модуля `cloud-provider-vsphere`

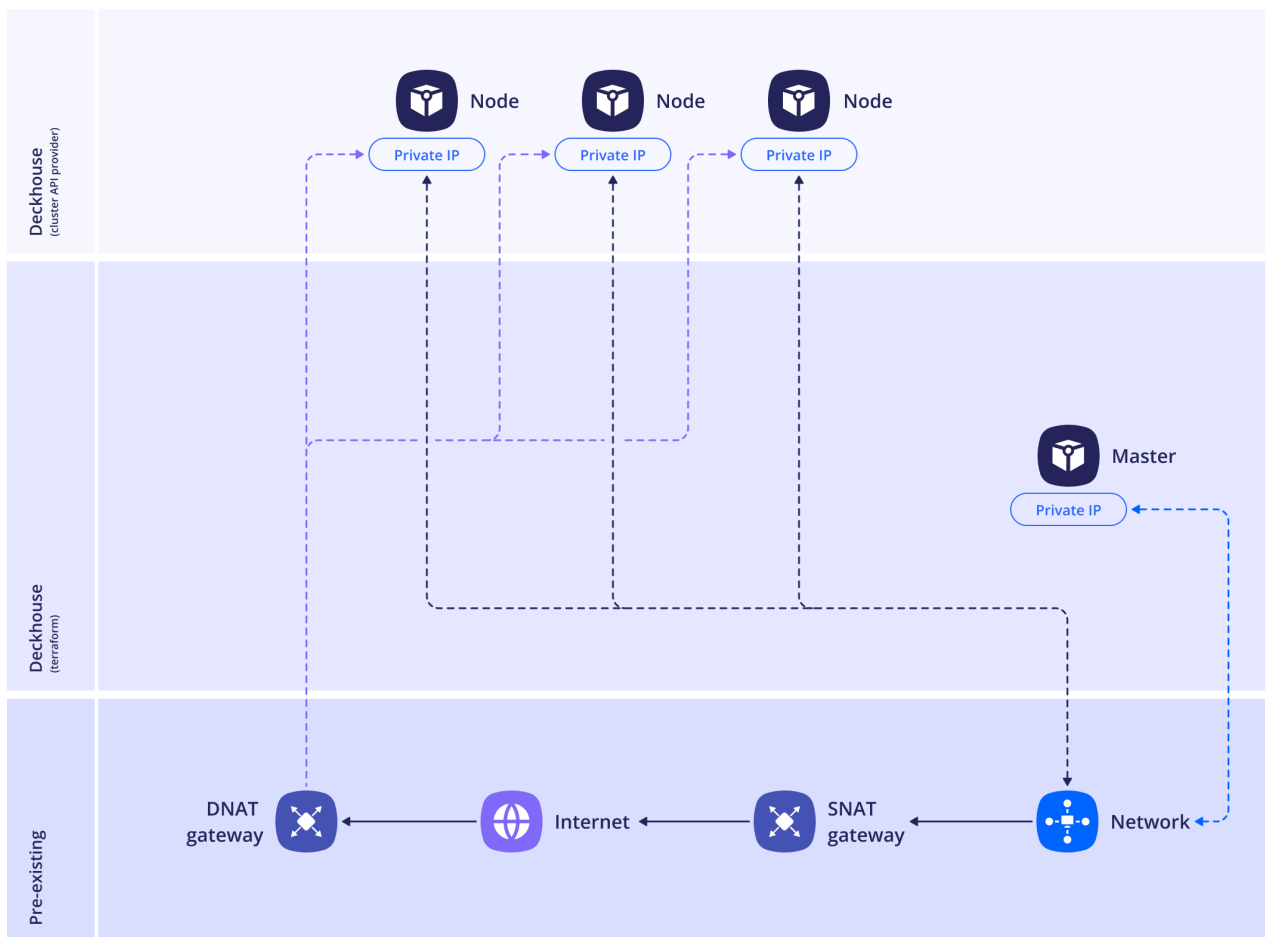
Схемы размещения и настройка VMware vSphere

Standard

Схема Standard предназначена для размещения кластера внутри инфраструктуры vSphere с возможностью управления ресурсами, сетями и хранилищем.

Особенности:

- Использование vSphere Datacenter в качестве региона (`region`);
- Использование vSphere Cluster в качестве зоны (`zone`);
- Поддержка нескольких зон и размещения узлов по зонам;
- Использование различных datastore для дисков и volume'ов;
- Поддержка подключения сетей, включая дополнительную сетевую изоляцию (например, MetalLB + BGP).



Пример конфигурации:

```

apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
    - ru-central1-a
    - ru-central1-b
instanceClass:
  numCPUs: 4
  memory: 8192
  template: dev/golden_image
  datastore: dev/lun_1
  mainNetwork: net3-k8s
nodeGroups:
  - name: khm
    replicas: 1
    zones:
      - ru-central1-a
    instanceClass:
      numCPUs: 4
      memory: 8192
      template: dev/golden_image
      datastore: dev/lun_1
      mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
  - ru-central1-a
  - ru-central1-b

```

Обязательные параметры ресурса VsphereClusterConfiguration:

- `region` — тег, присвоенный объекту Datacenter;
- `zoneTagCategory` и `regionTagCategory` — категории тегов, по которым распознаются регионы и зоны;
- `internalNetworkCIDR` — подсеть для назначения внутренних IP-адресов;
- `vmFolderPath` — путь до папки, в которой будут размещаться виртуальные машины кластера;
- `sshPublicKey` — публичный SSH-ключ для доступа к узлам;
- `zones` — список зон, доступных для размещения узлов.

i Все узлы, размещённые в разных зонах, должны иметь доступ к общим datastore с аналогичными тегами зоны.

Список необходимых привилегий

i Подробнее о том, как создать и назначить роль пользователю можно ознакомиться в разделах «Настройка через vSphere Client» и «Настройка через govc».

Детальный список привилегий, необходимых для работы Deckhouse Kubernetes Platform в vSphere:

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
—	— (назначаются по умолчанию при создании роли)	System.Anonymous System.Read System.View	Базовый доступ к объектам vSphere Inventory, необходимый для работы всех компонентов интеграции Deckhouse с vSphere.

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
Cns	Searchable	Cns.Searchable	Поиск и сопоставление объектов Container Native Storage при работе CSI-драйвера с томами Kubernetes.
Datastore	Allocate space, Browse datastore, Low level file operations	Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement	Выделение дисков при создании виртуальных машин и заказе PersistentVolumes в кластере.
Folder	Create folder, Delete folder, Move folder, Rename folder	Folder.Create Folder.Delete Folder.Move Folder.Rename	Группировка кластера Deckhouse Kubernetes Platform в одном Folder в vSphere Inventory.
Global	Global tag, System tag	Global.GlobalTag Global.SystemTag	Доступ к глобальным и системным тегам, используемым Deckhouse Kubernetes Platform при работе с объектами vSphere.
vSphere Tagging	Assign or Unassign vSphere Tag, Assign or Unassign vSphere Tag on Object, Create vSphere Tag, Create vSphere Tag Category, Delete vSphere Tag, Delete vSphere Tag Category, Edit vSphere Tag, Edit vSphere Tag Category, Modify UsedBy Field for Category, Modify UsedBy Field for Tag	InventoryService.Tagging.AttachTag InventoryService.Tagging.ObjectAttachable InventoryService.Tagging.CreateTag InventoryService.Tagging.CreateCategory InventoryService.Tagging.DeleteTag InventoryService.Tagging.DeleteCategory InventoryService.Tagging.EditTag InventoryService.Tagging.EditCategory InventoryService.Tagging.ModifyUsedByForCategory InventoryService.Tagging.ModifyUsedByForTag	Deckhouse Kubernetes Platform использует теги для определения доступных ему объектов Datacenter, Cluster и Datastore, а также для определения виртуальных машин, находящихся под его управлением.
Network	Assign network	Network.Assign	Подключение сетей и port group к виртуальным машинам кластера Deckhouse Kubernetes Platform.
Resource	Assign virtual machine to resource pool, Create resource pool, Modify resource pool, Remove resource pool, Rename resource pool	Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool	Размещение виртуальных машин кластера Deckhouse Kubernetes Platform в целевом пуле ресурсов и управление этим пулом.
VM Storage Policies (<i>Profile-driven Storage Privileges</i> в vSphere 7)	View VM storage policies (<i>Profile-driven storage view</i> в vSphere 7)	StorageProfile.View	Просмотр политик хранения, используемых при создании виртуальных машин и динамическом заказе томов в кластере.
vApp	Add virtual machine, Assign resource pool, Create, Delete, Import, Power Off, Power On, View OVF Environment, vApp application configuration, vApp instance configuration, vApp resource configuration	VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig	Управление операциями, связанными с развертыванием и конфигурацией vApp и OVF-шаблонов, используемых при создании виртуальных машин.

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
Virtual Machine > Change Configuration	Add existing disk, Add new disk, Add or remove device, Advanced configuration, Set annotation, Change CPU count, Toggle disk change tracking, Extend virtual disk, Acquire disk lease, Modify device settings, Configure managedBy, Change Memory, Query unowned files, Configure Raw device, Reload from path, Remove disk, Rename, Reset guest information, Change resource, Change Settings, Change Swapfile placement, Upgrade virtual machine compatibility	VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.RemoveDevice VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement VirtualMachine.Config.UpgradeVirtualHardware	Управление жизненным циклом виртуальных машин кластера Deckhouse Kubernetes Platform.
Virtual Machine > Edit Inventory	Create new, Create from existing, Remove, Move	VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete VirtualMachine.Inventory.Move	Создание, удаление и перемещение виртуальных машин кластера Deckhouse Kubernetes Platform в инвентаре vSphere.
Virtual Machine > Guest Operations	Guest Operation Queries	VirtualMachine.GuestOperations.Query	Получение информации из гостевой операционной системы виртуальных машин.
Virtual Machine > Interaction	Answer question, Device connection, Guest operating system management by VIX API, Power Off, Power On, Reset, Configure CD media, Install VMware Tools	VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall	Управление состоянием виртуальных машин, подключением устройств и взаимодействием с гостевой операционной системой.

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
Virtual Machine > Provisioning	Clone virtual machine, Customize guest, Deploy template, Allow virtual machine download, Allow virtual machine files upload, Read customization specifications	VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs	Клонирование шаблонов виртуальных машин, их настройка и развертывание при создании узлов кластера Deckhouse Kubernetes Platform.
Virtual Machine > Snapshot Management	Create snapshot, Remove Snapshot, Rename Snapshot	VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot	Управление снимками виртуальных машин и томов в сценариях, где эта функциональность используется компонентами платформы.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-vsphere`

Хранилище и балансировка нагрузки в VMware vSphere

Хранилище

Для хранения данных Kubernetes-кластера в VMware vSphere используются:

- Datastore — для размещения root-дисков виртуальных машин;
- CNS-диски (Container Native Storage) — для автоматического создания PersistentVolume'ов через CSI.

Deckhouse Kubernetes Platform автоматически создаёт StorageClass для каждого Datastore и DatastoreCluster, маркированных как `zone`.

Можно указать:

- имя StorageClass по умолчанию (`default`);
- исключения через `exclude` — список имен или шаблонов StorageClass, которые не нужно создавать.

Пример настройки через ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    storageClass:
      default: fast-lun102
      exclude:
        - ".*-lun101-.*"
        - slow-lun103
```

Изменение размера тома (PVC)

Deckhouse Kubernetes Platform поддерживает Online Resize PersistentVolume, начиная с версии vSphere 7.0U2.

Однако из-за особенностей CSI и API vSphere после изменения размера PVC требуется выполнить дополнительные действия:

1. Выполните `d8 k cordon <имя_узла>`.
2. Удалите под, использующий PVC.
3. Дождитесь завершения операции Resize:
 - убедитесь, что у PVC нет condition `Resizing`;
 - `FileSystemResizePending` можно игнорировать.
4. Выполните `d8 k uncordon <имя_узла>`.

Балансировка нагрузки

Варианты организации балансировки входящего трафика:

1. **Через внешний балансировщик.** Если в инфраструктуре уже есть внешний балансировщик (например, NSX-T), можно направлять трафик напрямую на frontend-узлы кластера.
2. **Через MetalLB.** Для отказоустойчивой балансировки внутри кластера рекомендуется использовать MetalLB в режиме BGP. В этом случае:

- frontend-узлы получают два сетевых интерфейса;
- требуется отдельный VLAN для BGP-трафика;
- необходим DHCP и доступ в интернет в этой сети;
- указываются IP-адреса и ASN BGP-роутеров;
- задаётся пул IP-адресов, который будет анонсироваться.



Необходимо обеспечить связь между BGP-роутерами и frontend-узлами в выделенном VLAN.

CSI

Подсистема хранения по умолчанию использует CNS-диски с возможностью изменения их размера на лету. Но также поддерживается работа и в legacy-режиме с использованием FCD-дисков. Поведение подсистемы устанавливается с помощью параметра `compatibilityFlag`.

Важная информация об увеличении размера PVC

Из-за особенностей работы volume-resizer CSI и vSphere API, после увеличения размера PVC нужно сделать следующее:

1. На узле, где находится под, выполните команду `d8 k cordon <имя_узла>`.
2. Удалите под.
3. Убедитесь, что изменение размера прошло успешно. В объекте PVC *не будет* condition `Resizing`.



Состояние `FileSystemResizePending` не является проблемой.

4. На узле, где находится под, выполните команду `d8 k uncordon <имя_узла>`.

Настройка Datastore



Разметку **Datastore** также можно сделать через **vSphere Client** по инструкции «Настройка через vSphere Client». Ниже описана настройка через **govc**.

Для корректной работы PersistentVolume необходимо, чтобы datastore был доступен на всех ESXi.

Назначьте теги:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Дополнительные ресурсы

- Параметры модуля `cloud-provider-vsphere`

Интеграция со службами VMware vSphere

Deckhouse Kubernetes Platform интегрируется с инфраструктурой VMware vSphere и использует ресурсы `VsphereInstanceClass` для описания характеристик виртуальных машин, создаваемых в составе кластера Kubernetes.

Основные возможности:

- Заказ и удаление виртуальных машин через vCenter API;
- Размещение узлов кластера в разных кластерах (`zones`) и датацентрах (`region`);
- Использование шаблонов виртуальных машин с `cloud-init`;
- Поддержка сетей с DHCP, статической адресацией и дополнительными интерфейсами;
- Работа с хранилищем: заказ root-дисков и PVC на базе Datastore или CNS-дисков;
- Поддержка механизмов балансировки входящего трафика:
 - через внешние балансировщики;
 - через MetalLB (в режиме BGP).



DKP поддерживает гибридную интеграцию с VMware vSphere. Подробнее о настройке можно узнать в разделе «Гибридный кластер с vSphere».

Управление ресурсами vSphere

Удаление CloudPermanent-узлов в vSphere

Узлы типа `CloudPermanent` создаются на основании конфигурации групп узлов, заданной в секции `nodeGroups` ресурса `VsphereClusterConfiguration`.

Параметр `replicas` определяет необходимое количество виртуальных машин в группе. После изменения

конфигурации необходимо выполнить команду `dhctl converge` для запуска Terraform и актуализации состояния виртуальных машин в VMware vSphere в соответствии с указанным количеством реплик.

Чтобы уменьшить количество узлов в группе, уменьшите значение `replicas` и выполните `dhctl converge`.



Не удаляйте описание группы из секции `nodeGroups`, пока значение параметра `replicas` больше нуля.

Если удалить описание группы до уменьшения количества реплик до нуля, состояние Terraform может рассинхронизироваться с состоянием виртуальных машин и дисков в VMware vSphere. В результате последующее выполнение `dhctl converge` может завершиться ошибкой и потребовать ручного восстановления состояния.

Уменьшение количества узлов

Чтобы уменьшить количество узлов в группе `CloudPermanent`:

1. Откройте конфигурацию vSphere для редактирования:

```
d8 system edit provider-cluster-configuration
```

2. В секции `nodeGroups` найдите необходимую группу и уменьшите значение параметра `replicas` до требуемого количества узлов.

Например, чтобы уменьшить количество узлов в группе `worker` с трёх до двух, установите `replicas: 2`:

```
nodeGroups:
- name: worker
  replicas: 2
  zones:
  - zone-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-24.04
    datastore: datastore-1
    mainNetwork: network-1
```

Сохраните изменения.

3. В установочном контейнере DKP примените изменённую конфигурацию:

```
dhctl converge \
--ssh-host <IP-АДРЕС_MASTER-УЗЛА> \
--ssh-user <ИМЯ_ПОЛЬЗОВАТЕЛЯ> \
--ssh-agent-private-keys /tmp/.ssh/<ИМЯ_ПРИВАТНОГО_SSH-КЛЮЧА>
```



Используйте установочный контейнер той же редакции и версии, что и в кластере.

4. Дождитесь завершения `dhctl converge` и проверьте количество узлов в группе:

```
d8 k get nodes -l node.deckhouse.io/group=<ИМЯ_ГРУППЫ>
```

Полное удаление группы узлов

Полное удаление группы `CloudPermanent` выполняется в два этапа. Сначала необходимо уменьшить количество реплик до нуля и дождаться удаления узлов и виртуальных машин. После этого можно удалить описание группы из конфигурации vSphere.

Чтобы полностью удалить группу `CloudPermanent`:

1. Откройте конфигурацию vSphere для редактирования:

```
d8 system edit provider-cluster-configuration
```

2. В секции `nodeGroups` найдите удаляемую группу и установите для неё значение `replicas: 0`. Не удаляйте описание группы на этом этапе. Например:

```
nodeGroups:
- name: worker
  replicas: 0
  zones:
  - zone-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-24.04
    datastore: datastore-1
    mainNetwork: network-1
```

Сохраните изменения.

- В установочном контейнере DKP примените изменённую конфигурацию:

```
dhctl converge \
--ssh-host <IP-АДРЕС_MASTER-УЗЛА> \
--ssh-user <ИМЯ_ПОЛЬЗОВАТЕЛЯ> \
--ssh-agent-private-keys /tmp/.ssh/<ИМЯ_ПРИВАТНОГО_SSH-КЛЮЧА>
```

- Дождитесь успешного завершения `dhctl converge` и убедитесь, что в группе не осталось узлов:

```
d8 k get nodes -l node.deckhouse.io/group=<ИМЯ_ГРУППЫ>
```

Команда не должна возвращать узлы удаляемой группы.

Также убедитесь с помощью vSphere Client, что связанные с группой виртуальные машины удалены.

- Повторно откройте конфигурацию vSphere:

```
d8 system edit provider-cluster-configuration
```

- Удалите описание группы из секции `nodeGroups` и сохраните изменения.

- Повторно примените конфигурацию:

```
dhctl converge \
--ssh-host <IP-АДРЕС_MASTER-УЗЛА> \
--ssh-user <ИМЯ_ПОЛЬЗОВАТЕЛЯ> \
--ssh-agent-private-keys /tmp/.ssh/<ИМЯ_ПРИВАТНОГО_SSH-КЛЮЧА>
```

- Убедитесь, что объект NodeGroup удалён:

```
d8 k get nodegroup <ИМЯ_ГРУППЫ>
```

Команда должна завершиться сообщением о том, что запрашиваемый ресурс не найден.

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-vsphere`
- Настройки провайдера модуля `cloud-provider-vsphere`

Подключение и авторизация в zVirt

Требования

Для корректной работы Deckhouse Kubernetes Platform с облаком zVirt необходимы:

- Рабочая инсталляция zVirt с доступным API;
- Версия zVirt 4.0-4.4;
- Пользователь с правами доступа к API и управлению виртуальными машинами;
- Хранилище с загруженным cloud-образом операционной системы;
- Подготовленный шаблон виртуальной машины на основе cloud-образа.



Облачный провайдер zVirt находится в экспериментальном статусе. Возможны изменения API и логики взаимодействия в будущих версиях.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Подготовка cloud-образа

1. Перейдите в раздел «zVirt → Хранилище → Диски».
2. Загрузите cloud-образ ОС (например, `.qcow2` или `.img`) в хранилище.

Новый Изменить Удалить Переместить Копировать Загрузить ▾ Скачать ⋮

Начать
Отменить
Приостановить
Возобновить

Загрузить образ ✕

Выберите файл `alse-vanilla-1.7.5-cloud-base-mg12.1.2.qcow2`

Формат: QCOW2 Содержимое: Данные
Размер: < 1 GiB QCOW2 Compat: 1.1
Виртуальный размер: 2 GiB Файл резервной копии: Нет

Настройки диска

Размер (GiB): 3 ☐ Очистить после удаления
Имя: Astra-Linux-Special-Edition-1.7.5 ☐ Может быть общим
Описание: Astra Linux cloud image ☒ Включить инкрементное резервное копирование
Центр данных: Default
Домен хранения: Local (1639 GiB свободно из 1687 GiB)
Профиль диска: Local
Хост: zvirt.local

Тест соединения

✓ Тест соединения с ovirt-imageio-proxy прошел успешно.

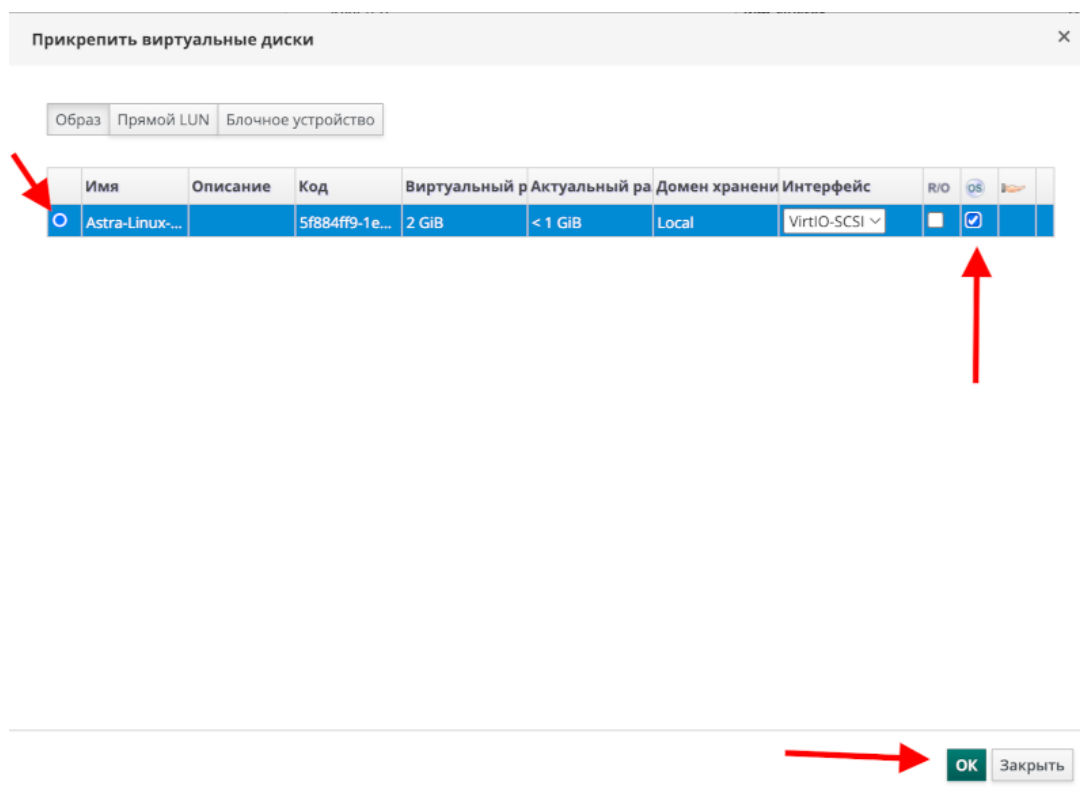
ОК Закреть

3. Дождитесь окончания загрузки. Статус диска должен быть `OK`.

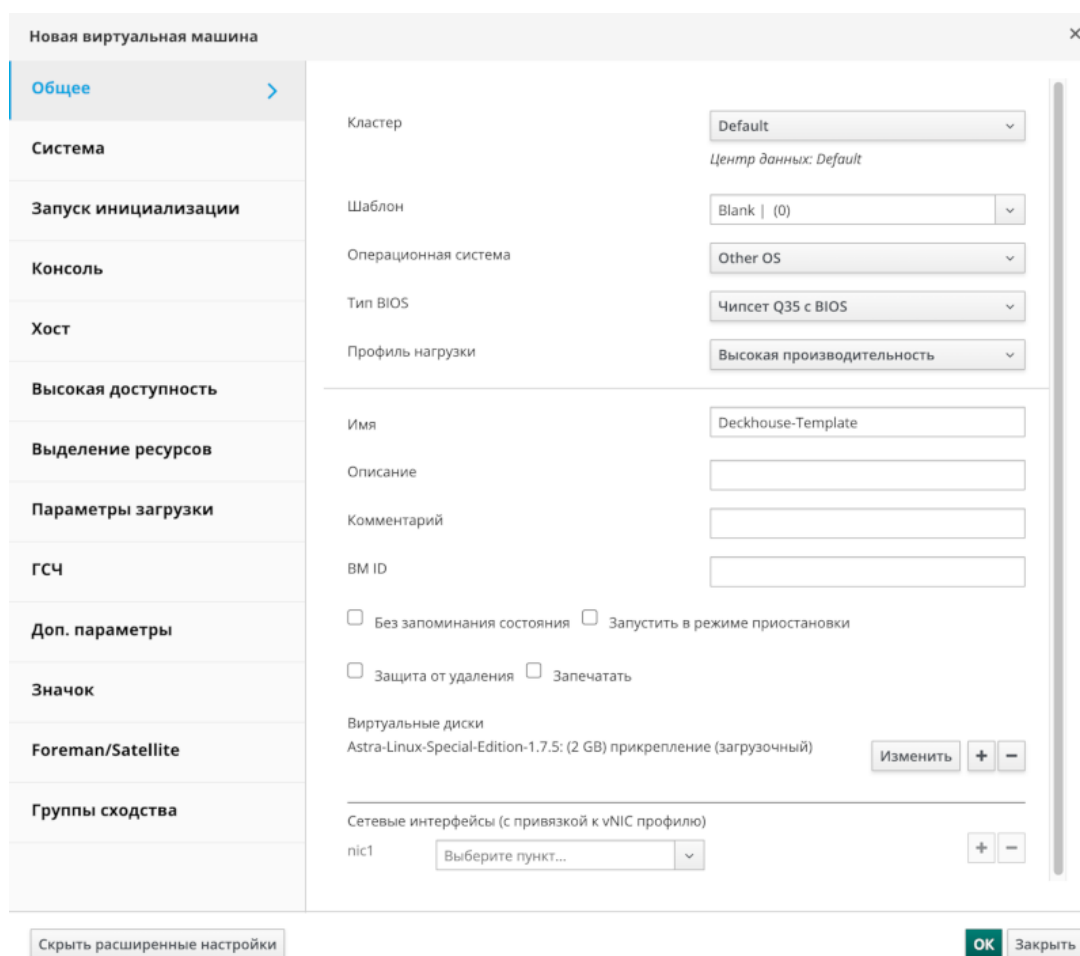
Имя	Идентификатор	Путь к файлу	Размер	Статус	Тип	Скорость	Время	Описание	Метрики	Метрики	Метрики
Astra-Linux-Special-Edition-1.7.5	70774afc-b4db-466e-ac...	alse-vanilla-1.7.5-cloud-base-mg12.1.2.qcow2	2 GiB	Загружено 132 из 591 MB	Образ	0MB/s	0MB/s				

Подготовка шаблона виртуальной машины

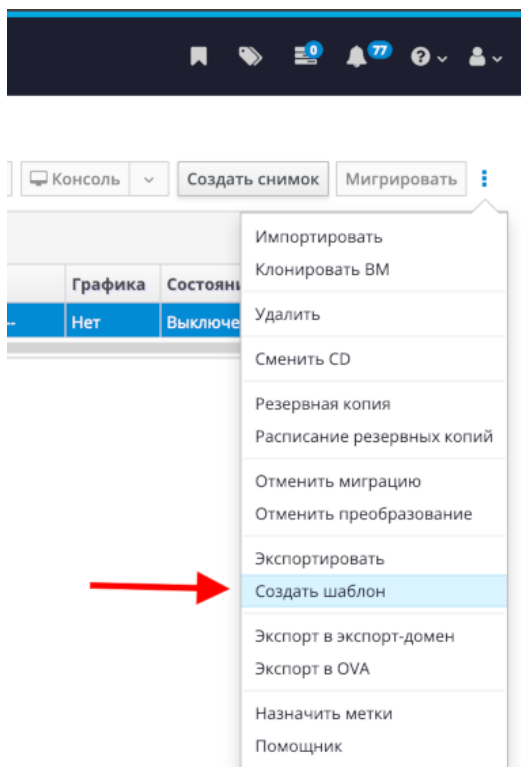
1. Откройте «Ресурсы → Виртуальные машины» и создайте новую VM.
2. Укажите параметры:
 - Шаблон: Blank;
 - Операционная система: соответствует загруженному cloud-образу;
 - Профиль нагрузки: высокая производительность;
 - Диски: прикрепите загруженный cloud-образ и установите как загрузочный.



3. Не добавляйте сетевые интерфейсы — они будут созданы автоматически при заказе узла Deckhouse.



4. Сохраните и создайте виртуальную машину.
5. После создания откройте карточку VM и создайте на её основе шаблон.



6. Укажите имя шаблона и включите опцию «Запечатать шаблон».

Новый шаблон [X]

Имя: (red arrow points here)

Описание:

Комментарий:

Кластер:

Профиль ЦП:

☐ Создать, как подверсию шаблона

Выделение дискового пространства:

Имя	Виртуальный размер	Формат	Цель	Профиль диска
Astra-Linux-Special-Edi	2 GiB	QCOW2	Local (1638 GiB свг	Local

☒ Разрешить всем пользователям доступ к шаблону

☐ Копировать разрешения VM

☒ Запечатать шаблон (только Linux) (red arrow points here)

[OK] [Закрыть] (red arrow points to OK button)

7. Убедитесь, что шаблон успешно создан в разделе «Ресурсы → Шаблоны».

Имя	Версия	Комментарий	Дата создания	Состояние	Запечатано
Deckhouse-Node			14 февр. 2024 г., 16:23:...	OK	Да

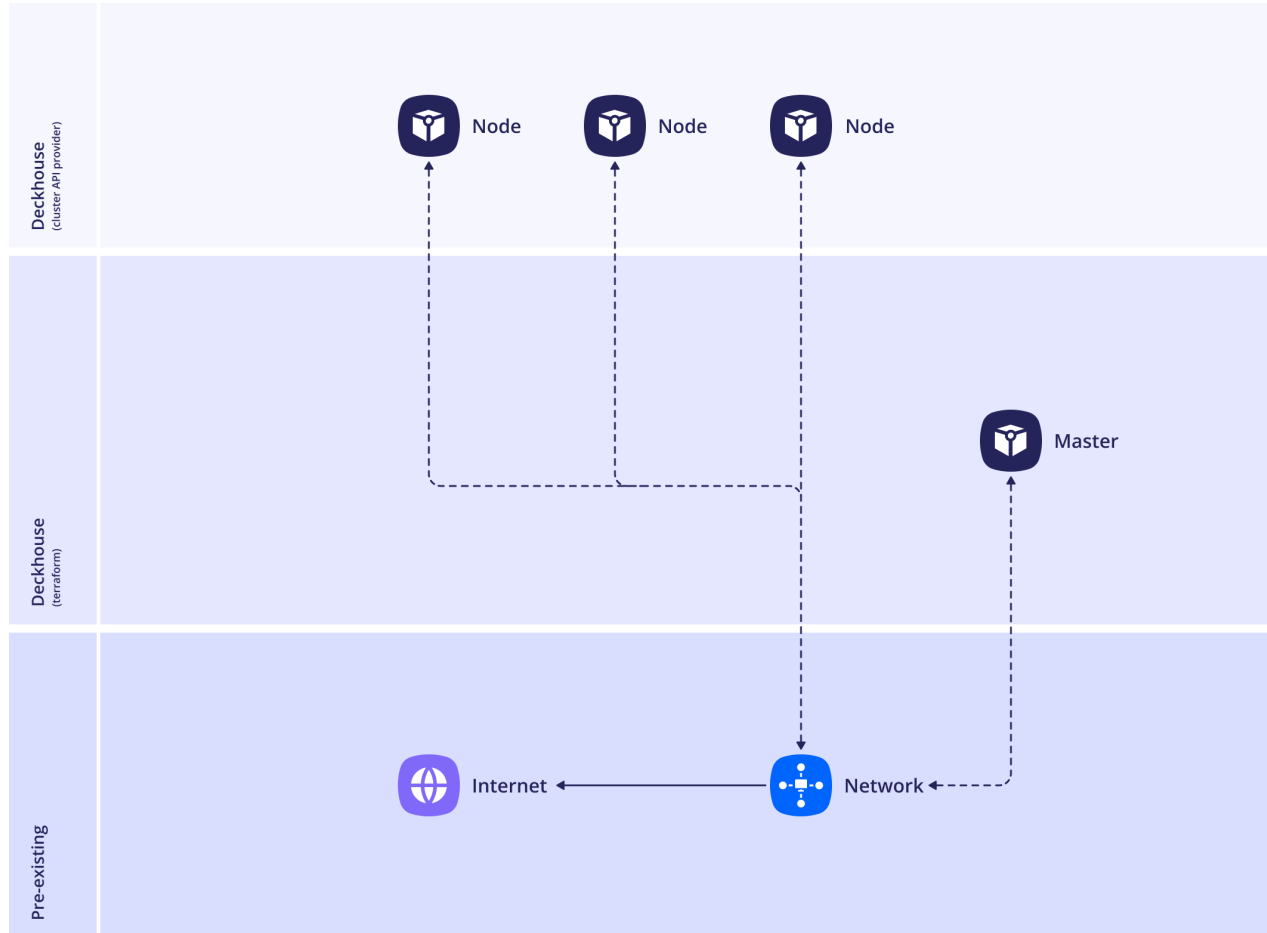
Схемы размещения и настройка zVirt

Standard

Схема размещения Standard используется для интеграции Deckhouse Kubernetes Platform с виртуальной инфраструктурой zVirt. Эта схема предполагает развертывание всех узлов в пределах одного кластера zVirt с централизованным управлением шаблонами, хранилищем и сетями.

Особенности:

- Использование одного zVirt-кластера (`clusterID`);
- Хранилище (Storage Domain), доступное всем хостам кластера;
- Использование шаблона виртуальной машины, созданного из cloud-образа;
- Присвоение сетевого профиля vNIC при заказе ВМ;
- Полная автоматизация создания и удаления узлов через API zVirt.



Пример конфигурации:

```
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
clusterID: "b46372e7-0d52-40c7-9bbf-fda31e187088"
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    rootDiskSizeGb: 40
    template: ALT-p10
    vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
    storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"
```

Обязательные параметры ресурса ZvirtClusterConfiguration:

- `clusterID` — UUID кластера в zVirt, где размещаются виртуальные машины;
- `sshPublicKey` — публичный SSH-ключ для доступа на узлы;
- `template` — имя подготовленного шаблона виртуальной машины;
- `vnicProfileID` — UUID сетевого профиля (vNIC);
- `storageDomainID` — UUID хранилища (Storage Domain), в котором размещаются диски.

i Значения UUID (`clusterID`, `vnicProfileID`, `storageDomainID`) можно получить через API zVirt или интерфейс администратора.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-zvirt`

Хранилище и балансировка нагрузки в zVirt

Хранилище

В кластере, размещённом в инфраструктуре zVirt, используются хранилища (Storage Domain), доступные в пределах заданного `clusterID`. Все диски виртуальных машин создаются внутри указанного хранилища.

Требования

- Указанный в конфигурации `storageDomainID` должен быть доступен для `clusterID`, заданного в `ZvirtClusterConfiguration`;
- Диск будет создан на основе шаблона (`template`) и размещён в этом домене хранения;
- При заказе `PersistentVolume` используются root-диски машин — отдельные PVC в zVirt пока не поддерживаются.

Конфигурация

Фрагмент `ZvirtClusterConfiguration` с указанием домена хранения:

```
masterNodeGroup:
  replicas: 1
instanceClass:
  numCPUs: 4
  memory: 8192
  rootDiskSizeGb: 40
  template: ALT-p10
  vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
  storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"
```

i Используйте уникальные идентификаторы (UUID) для указания шаблона и хранилища. Получить их можно через zVirt API или интерфейс управления.

Балансировка нагрузки

Платформа zVirt не предоставляет встроенного балансировщика нагрузки. Для организации входящего трафика рекомендуются следующие подходы:

1. Использование внешнего балансировщика. Если в вашей инфраструктуре уже есть внешний балансировщик (например, аппаратный или программный), настройте проброс трафика (80/443) на frontend-узлы кластера.
2. Использование MetalLB. Для обеспечения отказоустойчивой балансировки в кластере можно использовать MetalLB в L2-режиме.

Рекомендации:

- Выделите отдельную L2-сеть с DHCP и интернет-доступом;
- Настройте диапазон IP-адресов, из которого MetalLB будет анонсировать адреса;
- Поддержите подключение этой сети к frontend-узлам кластера;
- В конфигурации `VirtualMachineTemplate` оставьте конфигурацию сетевых интерфейсов пустой — Deckhouse Kubernetes Platform сам добавит их на этапе запуска VM.

i Поддержка BGP-режима MetalLB в zVirt не гарантируется и зависит от сетевой инфраструктуры.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-zvirt`

Интеграция со службами zVirt

i Интеграция с zVirt находится в статусе экспериментальной. Возможны изменения в интерфейсах и функциональности.

Deckhouse Kubernetes Platform поддерживает интеграцию с инфраструктурой zVirt и позволяет заказывать, управлять и удалять виртуальные машины с помощью описаний в ресурсе `ZvirtInstanceClass`.

Основные возможности

- Заказ виртуальных машин в zVirt при создании или масштабировании NodeGroup.
- Использование заранее подготовленных шаблонов виртуальных машин (template).
- Автоматическая настройка сетевых интерфейсов и подключение к кластеру.
- Поддержка динамического и статического выделения IP-адресов.
- Размещение дисков на заданном storage domain.
- Поддержка cloud-образов с `cloud-init` для корректной инициализации узлов.

Дополнительные ресурсы

- Кастомные ресурсы модуля `cloud-provider-zvirt`

Гибридные кластеры

Гибридный кластер — это статический кластер Deckhouse Kubernetes Platform (DKP), расширенный за счёт узлов, размещённых в инфраструктуре поддерживаемого провайдера, например Yandex Cloud, VMware Cloud Director или VMware vSphere.

В таком сценарии control plane и исходные узлы кластера остаются частью статического кластера DKP, а дополнительные узлы добавляются через модуль соответствующего провайдера. Эти узлы могут создаваться автоматически через API провайдера или подключаться вручную, если виртуальные машины были созданы заранее.

Такой подход позволяет расширять существующий статический кластер без создания отдельного Kubernetes-кластера: увеличивать вычислительные мощности, размещать часть рабочих нагрузок в другой инфраструктуре или постепенно переносить туда сервисы. Для приложений при этом сохраняется единая плоскость управления Kubernetes: общий API, единые ресурсы, единые механизмы планирования, мониторинга, обновления и эксплуатации.

Гибридная интеграция выполняется на базе уже развёрнутого статического кластера DKP (`clusterType: Static`). Затем включается модуль соответствующего облачного провайдера, через который DKP получает информацию о внешней инфраструктуре и может создавать или подключать дополнительные узлы.

В зависимости от провайдера узлы можно добавлять автоматически через API провайдера или подключать заранее созданные виртуальные машины вручную через bootstrap-скрипт.

Общие принципы гибридной интеграции описаны на этой странице. Порядок подготовки инфраструктуры и добавления узлов зависит от используемого провайдера и приведён в отдельных инструкциях:

- Yandex Cloud;
- VMware Cloud Director;
- VMware vSphere;

Типы групп узлов

В гибридном сценарии DKP используются следующие типы групп узлов:

- `CloudEphemeral` — узлы, которые DKP создаёт и удаляет автоматически через API настроенного облачного провайдера.
- `CloudStatic` — узлы, которые создаются пользователем вручную или внешними инструментами в той же облачной инфраструктуре, с которой настроена интеграция у облачного провайдера. На таких узлах работает CSI, а объект Node управляется cloud-controller-manager: он автоматически обогащается информацией о зоне и регионе по данным провайдера.

Способы добавления узлов

Подключение узлов из инфраструктуры провайдера может выполняться следующими способами:

- **Автоматическое создание облачных узлов.** DKP создаёт виртуальные машины через API провайдера. Параметры VM описываются ресурсом `*InstanceClass`, а требуемое количество узлов и зоны размещения — ресурсом NodeGroup с типом `CloudEphemeral`.
- **Подключение вручную созданных облачных узлов через bootstrap-скрипт.** Виртуальная машина создаётся пользователем заранее в инфраструктуре провайдера, после чего на ней запускается bootstrap-скрипт DKP для подключения узла к кластеру. Для такого сценария используется NodeGroup с типом `CloudStatic`. Такой узел управляется cloud-controller-manager соответствующего провайдера.

В этом разделе описаны общие сетевые требования для гибридных кластеров. Требования, связанные с конкретными провайдерами, включая сетевые параметры, шаблоны виртуальных машин, учётные данные, параметры размещения и способы добавления узлов, приведены в отдельных инструкциях для Yandex Cloud, VCD и vSphere.

Общие сетевые требования

Между узлами исходного статического кластера и подключаемыми узлами, размещёнными в инфраструктуре провайдера, должна быть настроена двусторонняя сетевая связность, достаточная для работы компонентов DKP и Kubernetes.

Связность должна обеспечивать:

- доступ подключаемых узлов к Kubernetes API исходного кластера;
- доступ подключаемых узлов к DNS-серверам;
- доступ подключаемых узлов к хранилищу образов контейнеров и другим внешним сервисам, необходимым для загрузки образов и пакетов;
- доступ control plane и системных компонентов к подключаемым узлам для работы kubelet, сетевых компонентов, мониторинга и служебных операций Kubernetes;

- доступ компонентов DKP, взаимодействующих с инфраструктурой провайдера, к API соответствующего провайдера.

Полный перечень соединений приведён в разделе «Сетевое взаимодействие», а рекомендации по ограничениям доступа — в разделе «Настройка сетевых политик».

Дополнительно рекомендуется проверить:

- маршрутизацию между сетями исходного статического кластера и подключаемых узлов в обоих направлениях;
- одинаковое значение MTU на всём сетевом пути, особенно при использовании туннелей;
- параметры инкапсуляции трафика при использовании Cilium, включая `tunnelMode`, если между площадками применяется фильтрация трафика.

Конкретные требования к сетям, подсетям, шаблонам виртуальных машин, учётным данным и дополнительным параметрам зависят от используемого провайдера инфраструктуры и приведены в разделе «Предварительные требования» для соответствующего провайдера.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `node-manager`

Гибридный кластер с Yandex Cloud

Далее описан процесс добавления узлов из Yandex Cloud в существующий статический кластер Deckhouse Kubernetes Platform (DKP).

Для интеграции с Yandex Cloud используется модуль `cloud-provider-yandex`. Он обеспечивает взаимодействие DKP с API Yandex Cloud, получение информации об облачной инфраструктуре, создание виртуальных машин, работу с сетевыми параметрами и подключение узлов к существующему кластеру.

В разделе описаны два способа добавления узлов:

- **Автоматическое создание узлов в Yandex Cloud.** DKP создаёт виртуальные машины через API Yandex Cloud. Параметры ВМ задаются ресурсом `YandexInstanceClass`, а требуемое количество узлов и зоны размещения — ресурсом `NodeGroup` с типом `CloudEphemeral`.
- **Подключение вручную созданных узлов через bootstrap-скрипт.** Виртуальная машина создаётся пользователем заранее и подключается к кластеру с помощью bootstrap-скрипта DKP. Для такого сценария используется `NodeGroup` с типом `CloudStatic`.

Предварительные требования

Перед началом убедитесь, что выполнены следующие условия:

- Кластер создан с параметром `clusterType: Static`.
- Между сетью статических узлов и VPC Yandex Cloud настроена сетевая связность. Добавляемые узлы Yandex Cloud должны иметь доступ к Kubernetes API, DNS и необходимым адресам согласно разделам «Сетевое взаимодействие» и «Настройка сетевых политик». При использовании Cilium с туннелированием трафика подов должен быть выбран режим `tunnelMode`, соответствующий сетевой связности между площадками.
- Выполнены требования из раздела «Подключение и авторизация в Yandex Cloud»:
 - подготовлен сервисный аккаунт;
 - выбран каталог, в котором будут создаваться ресурсы;
 - настроены необходимые роли и доступ к используемой VPC.

Добавление автоматически создаваемых узлов

Для выполнения подготовительных команд нужен Yandex Cloud CLI (`yc`). Его можно использовать на рабочей машине администратора. На master-узле кластера `yc` не требуется: в кластере нужно применить только подготовленные манифесты.

1. Получите идентификаторы облака и каталога, в котором будут создаваться узлы:

```
yc resource-manager cloud list
yc resource-manager folder list
```

2. Укажите полученные идентификаторы в переменных:

```
export CLOUD_ID=<CLOUD_ID>
export FOLDER_ID=<FOLDER_ID>
```

Где:

- `CLOUD_ID` — ID облака Yandex Cloud;
- `FOLDER_ID` — ID каталога, в котором будут создаваться ресурсы.

3. Получите идентификаторы сети, подсети и зоны, где будут создаваться узлы:

```
yc vpc network list --folder-id "$FOLDER_ID"
yc vpc subnet list --folder-id "$FOLDER_ID"
yc compute zone list
```

4. Укажите полученные значения в переменных:

```
export NETWORK_ID="<<NETWORK_ID>"
export SUBNET_ID="<<SUBNET_ID>"
export ZONE="<<ZONE>"
```

Где:

- `NETWORK_ID` — ID VPC-сети;
- `SUBNET_ID` — ID подсети, в которой будут создаваться узлы;
- `ZONE` — зона доступности, соответствующая выбранной подсети, например `ru-central1-a`.

Подробнее в разделе «Подключение и авторизация в Yandex Cloud».

5. Создайте сервисный аккаунт в нужном каталоге Yandex Cloud и назначьте ему права:

```
yc iam service-account create \
  --name dkp-hybrid \
  --folder-id "$FOLDER_ID"

export SA_ID="$(yc iam service-account get \
  --name dkp-hybrid \
  --folder-id "$FOLDER_ID" \
  --format json | jq -r .id)"

yc resource-manager folder add-access-binding "$FOLDER_ID" \
  --role editor \
  --subject "serviceAccount:${SA_ID}"

yc resource-manager folder add-access-binding "$FOLDER_ID" \
  --role vpc.admin \
  --subject "serviceAccount:${SA_ID}"
```

Роль `editor` нужна для создания и управления облачными ресурсами, а `vpc.admin` — для работы с сетевыми ресурсами VPC.

6. Создайте ключ сервисного аккаунта и сохраните его в JSON-файл:

```
yc iam key create \
  --service-account-id "$SA_ID" \
  --output dkp-hybrid-sa-key.json
```

Сохраните JSON-ключ сервисного аккаунта в переменную окружения `SERVICE_ACCOUNT_JSON` в однострочном формате:

```
export SERVICE_ACCOUNT_JSON="$(jq -c . dkp-hybrid-sa-key.json)"
```

7. Сохраните публичный SSH-ключ администратора в переменную окружения `SSH_PUBLIC_KEY`:

```
export SSH_PUBLIC_KEY="$(cat ~/.ssh/id_rsa.pub)"
```

Если используется другой ключ, укажите путь к нему вместо `~/.ssh/id_rsa.pub`.

⚠ В переменную `SSH_PUBLIC_KEY` нужно сохранить публичный SSH-ключ администратора, который будет использоваться для доступа к создаваемым узлам. Не используйте публичный ключ из JSON-файла сервисного аккаунта.

8. Получите ID образа операционной системы, из которого будут создаваться виртуальные машины, и сохраните его в переменную окружения `IMAGE_ID`:

```
export IMAGE_ID="$(yc compute image get-latest-from-family ubuntu-2404-lts \
  --folder-id standard-images \
  --format json | jq -r .id)"
```

⚠ В переменную `IMAGE_ID` нужно сохранить ID образа ОС в Yandex Cloud. Не используйте ID существующей виртуальной машины или ID ключа сервисного аккаунта.

9. Укажите CIDR сети, в которой будут размещаться узлы Yandex Cloud:

```
export NODE_NETWORK_CIDR="<<NODE_NETWORK_CIDR>"
```

`NODE_NETWORK_CIDR` — CIDR, включающий внутренние IP-адреса узлов Yandex Cloud. Для одной зоны обычно совпадает с CIDR выбранной подсети. Например, если узлы создаются в подсети `10.128.0.0/24`, укажите `10.128.0.0/24`. Узнать CIDR подсети можно командой:


```
yc vpc subnet list --folder-id "${FOLDER_ID}"
```

10. Создайте файл с конфигурацией провайдера. Например, `cloud-provider-cluster-configuration.yaml`:

```
cat > cloud-provider-cluster-configuration.yaml <<EOF
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: ${IMAGE_ID}
    diskSizeGB: 100
    platform: standard-v3
    externalIPAddresses:
      - "Auto"
nodeNetworkCIDR: ${NODE_NETWORK_CIDR}
existingNetworkID: empty
provider:
  cloudID: ${CLOUD_ID}
  folderID: ${FOLDER_ID}
  serviceAccountJSON: '${SERVICE_ACCOUNT_JSON}'
  sshPublicKey: '${SSH_PUBLIC_KEY}'
EOF
```

В манифест автоматически подставляются значения переменных окружения, заданных на предыдущих шагах: `CLOUD_ID`, `FOLDER_ID`, `IMAGE_ID`, `NODE_NETWORK_CIDR`, `SERVICE_ACCOUNT_JSON` и `SSH_PUBLIC_KEY`.

i В гибридном сценарии, когда control plane уже развёрнут как статический кластер, секция `masterNodeGroup` не приводит к созданию master-узлов в Yandex Cloud, но остаётся частью конфигурации провайдера.

11. Создайте файл с discovery-данными Yandex Cloud. Например, `cloud-provider-discovery-data.json`:

```
cat > cloud-provider-discovery-data.json <<EOF
{
  "apiVersion": "deckhouse.io/v1",
  "defaultLbTargetGroupNetworkId": "empty",
  "internalNetworkIDs": [
    "${NETWORK_ID}"
  ],
  "kind": "YandexCloudDiscoveryData",
  "monitoringAPIKey": "",
  "region": "ru-central1",
  "routeTableID": "empty",
  "shouldAssignPublicIPAddress": false,
  "zoneToSubnetIdMap": {
    "${ZONE}": "${SUBNET_ID}"
  },
  "zones": [
    "${ZONE}"
  ]
}
EOF
```

В файл автоматически подставляются значения переменных окружения, заданных на предыдущих шагах: `NETWORK_ID`, `SUBNET_ID` и `ZONE`.

Параметр `shouldAssignPublicIPAddress` управляет назначением публичных IP-адресов создаваемым узлам. В примере указано значение `false`, поэтому создаваемые узлы будут получать только внутренние IP-адреса.

⚠ Если `shouldAssignPublicIPAddress` установлен в `false`, создаваемые узлы должны иметь доступ к хранилищу образов и внешним сервисам через NAT Gateway, NAT-инстанс, прокси или другой egress-механизм. Для зон, в которых подсети отсутствуют, допустимо использовать значение `empty`.

12. Закодируйте файлы `cloud-provider-cluster-configuration.yaml` и `cloud-provider-discovery-data.json` в Base64:

```
export CLUSTER_CONFIGURATION_B64="$(base64 -w0 cloud-provider-cluster-configuration.yaml)"
export DISCOVERY_DATA_B64="$(base64 -w0 cloud-provider-discovery-data.json)"
```

13. Создайте манифест с секретом `d8-provider-cluster-configuration` и `ModuleConfig` для включения и настройки модуля `cloud-provider-yandex`:

```
cat > yandex-provider-secret-and-mc.yaml <<EOF
apiVersion: v1
kind: Secret
metadata:
  labels:
    name: d8-provider-cluster-configuration
    name: d8-provider-cluster-configuration
    namespace: kube-system
type: Opaque
data:
  cloud-provider-cluster-configuration.yaml: ${CLUSTER_CONFIGURATION_B64}
  cloud-provider-discovery-data.json: ${DISCOVERY_DATA_B64}
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      default: network-ssd
EOF
```

14. Скопируйте файл `yandex-provider-secret-and-mc.yaml` на master-узел кластера. Примените манифест:

```
d8 k apply -f yandex-provider-secret-and-mc.yaml
```

15. Дождитесь включения модуля `cloud-provider-yandex` и появления кастомного ресурса `YandexInstanceClass`:

```
d8 k get module cloud-provider-dvp -O wide
d8 k get crd yandexinstanceclasses.deckhouse.io
```

16. Создайте файл с манифестами `YandexInstanceClass` и `NodeGroup`. Например, `yandex-instanceclass-nodegroup.yaml`:

```
apiVersion: deckhouse.io/v1alpha1
kind: YandexInstanceClass
metadata:
  name: yc-worker
spec:
  cores: 4
  memory: 8192
  diskSizeGB: 50
  diskType: network-ssd
  mainSubnet: <SUBNET_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: yc-worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: YandexInstanceClass
      name: yc-worker
    minPerZone: 1
    maxPerZone: 1
    zones:
      - ru-central1-a
```

Где:

- `YandexInstanceClass` описывает параметры виртуальной машины, которая будет создана в Yandex Cloud;
- `mainSubnet` — ID подсети, из которой создаваемые узлы должны иметь доступ к статическим узлам

- кластера;
- o NodeGroup описывает группу узлов, которую DKP должен поддерживать в кластере;
- o `nodeType: CloudEphemeral` означает, что узлы будут создаваться автоматически через облачного провайдера;
- o `cloudInstances.zones` должен содержать зоны из списка `zones` в `cloud-provider-discovery-data.json`.

17. Примените манифест:

```
d8 k apply -f yandex-instanceclass-nodegroup.yaml
```

После применения DKP начнёт создавать виртуальную машину в Yandex Cloud через `machine-controller-manager`.

18. Проверьте появление узла в кластере:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL - IP
static-master-0	Ready	control-plane,master	1h	v1.33.10	10.128.0.15
yc-worker-f3564dca-7fc59-s2w5d	Ready	yc-worker	10m	v1.33.10	10.128.0.21

19. Для диагностики состояния и поиска возможных проблем проверьте логи `machine-controller-manager`:

```
d8 k -n d8-cloud-instance-manager get machinedeployments.machine.sapcloud.io -O wide
d8 k -n d8-cloud-instance-manager get machinesets.machine.sapcloud.io -O wide
d8 k -n d8-cloud-instance-manager get machines.machine.sapcloud.io -O wide
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Добавление вручную созданных узлов через bootstrap-скрипт

Перед началом убедитесь, что выполнены следующие условия:

- Модуль `cloud-provider-yandex` включён:

```
d8 k get module cloud-provider-yandex -O wide
```

- Компоненты модуля `cloud-provider-yandex` находятся в состоянии `Running`:

```
d8 k -n d8-cloud-provider-yandex get pods -O wide
```

- В Yandex Cloud создана виртуальная машина, которая будет подключена к кластеру.
- Виртуальная машина подключена к сети и подсети Yandex Cloud, используемым для гибридной интеграции с кластером.
- У виртуальной машины есть сетевой интерфейс в VPC-сети и подсети Yandex Cloud, используемые для гибридной интеграции с кластером. IP-адрес этого интерфейса должен входить в CIDR, указанный в `nodeNetworkCIDR`, и быть доступен со стороны статических узлов кластера.
- Имя виртуальной машины в Yandex Cloud совпадает с именем хоста (`hostname`) внутри операционной системы.
- На виртуальной машине установлен один из пакетных менеджеров (`apt/apt-get`, `yum` или `rpm`) для поддерживаемой ОС. В РЕД ОС по умолчанию могут отсутствовать `yum` и `which`, поэтому их необходимо заранее установить.

1. Проверьте метаданные виртуальной машины в Yandex Cloud.

В метаданных VM должен быть настроен `cloud-init` с пользователем, через которого будет выполняться подключение по SSH.

Пример метаданных:

```
#cloud-config
datasource:
  Ec2:
    strict_id: false
ssh_pwauth: no
users:
  - name: <USER>
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash
    ssh_authorized_keys:
      - <SSH_PUBLIC_KEY>
```

Где:

- `<USER>` — имя пользователя для SSH-доступа к виртуальной машине;
- `<SSH_PUBLIC_KEY>` — публичный SSH-ключ администратора.

- На master-узле создайте файл с манифестом NodeGroup, указав имя группы узлов (в примере здесь и далее используется имя `yc-manual`). Например, `yandex-manual-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: yc-manual
spec:
  nodeType: CloudStatic
```

- Убедитесь, что NodeGroup создана и синхронизирована:

```
d8 k get nodegroup yc-manual
d8 k describe nodegroup yc-manual
```

Пример ожидаемого результата:

NAME	TYPE	READY	NODES	UPTODATE	INSTANCES	DESIRED	MIN	MAX	STANDBY
STATUS	AGE	SYNCED							
yc-manual	CloudStatic	0	0	0					
1m	True								

- На master-узле получите bootstrap-скрипт для созданной NodeGroup:

```
NODE_GROUP=yc-manual

d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
-o jsonpath='{.data.bootstrap\.sh}' > ${NODE_GROUP}-bootstrap.b64
```

- Скопируйте bootstrap-скрипт на подключаемую VM. Если SSH-доступ к VM есть с master-узла, выполните на master-узле:

```
scp ${NODE_GROUP}-bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
```

i Для копирования и запуска bootstrap-скрипта используйте пользователя, указанного в метаданных VM.

Если SSH-доступ к VM есть только с рабочей машины администратора, сначала скопируйте файл с master-узла на рабочую машину, а затем с рабочей машины на VM:

```
scp <MASTER_USER>@<MASTER_IP>:/root/${NODE_GROUP}-bootstrap.b64 ./bootstrap.b64
scp ./bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
```

Где:

- `<MASTER_USER>` — пользователь для SSH-доступа к master-узлу;
- `<MASTER_IP>` — IP-адрес master-узла;
- `<USER>` — пользователь на подключаемой VM;
- `<NODE_PUBLIC_OR_INTERNAL_IP>` — публичный или внутренний IP-адрес подключаемой VM.

- На подключаемой VM декодируйте bootstrap-скрипт, назначьте права и запустите его:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

После запуска bootstrap-скрипт установит необходимые компоненты, настроит container runtime, kubelet и подключит узел к кластеру.

- На master-узле проверьте появление нового узла:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
static-master-0	Ready	master	1h	v1.33.10	10.128.0.15	<none>
yandex-worker-hybrid	Ready	yc-manual	5m	v1.33.10	10.128.0.17	<PUBLIC_IP>

8. При сбоях подключения проверьте состояние NodeGroup, события и логи компонентов:

```
d8 k get nodegroup yc-manual
d8 k describe nodegroup yc-manual
d8 k describe node yandex-worker-hybrid
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `cloud-provider-yandex`
- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `node-manager`

Гибридный кластер с VCD

Далее описан процесс добавления узлов из VMware Cloud Director (VCD) в существующий статический кластер Deckhouse Kubernetes Platform (DKP).

Для интеграции с VCD используется модуль `cloud-provider-vcd`. Он обеспечивает взаимодействие DKP с VMware Cloud Director, создание и удаление виртуальных машин, получение информации об инфраструктуре VCD, а также интеграцию со StorageClass и другими возможностями провайдера.

В разделе описаны два способа добавления узлов:

- **Автоматическое создание узлов в VCD.** DKP создаёт виртуальные машины через API VCD. Параметры VM задаются ресурсом `VCDInstanceClass`, а требуемое количество узлов — ресурсом `NodeGroup` с типом `CloudEphemeral`.
- **Подключение вручную созданных узлов через bootstrap-скрипт.** Виртуальная машина создаётся пользователем заранее и подключается к кластеру с помощью bootstrap-скрипта DKP. Для такого сценария используется `NodeGroup` с типом `CloudStatic`.

Предварительные требования для VCD

Перед началом убедитесь, что выполнены следующие условия:

- Кластер создан с параметром `clusterType: Static`.
- Между сетью статических узлов и сетью виртуальных машин в VCD настроена сетевая связность. Узлы VCD, добавляемые в кластер, имеют доступ к Kubernetes API, DNS и необходимым адресам согласно разделам «Сетевое взаимодействие» и «Настройка сетевых политик». При использовании Cilium с туннелированием трафика подов выбран режим `tunnelMode`, соответствующий сетевой связности между площадками.
- Выполнены требования из раздела «Подключение и авторизация в VMware vCloud Director»:
 - настроен тенант в VCD с выделенными ресурсами;
 - подготовлена учётная запись VCD со статичным паролем и правами администратора;
 - настроена рабочая сеть в VCD с включённым DHCP-сервером;
 - подготовлены необходимые ресурсы VCD: VDC, vApp, шаблоны, политики и другие параметры.

Добавление автоматически создаваемых узлов

1. Создайте файл с ресурсом `ModuleConfig`. Например, `cloud-provider-vcd-mc.yaml`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vcd
spec:
  version: 1
  enabled: true
  settings:
    mainNetwork: <NETWORK_NAME>
    organization: <ORGANIZATION>
    virtualDataCenter: <VDC_NAME>
    virtualApplicationName: <VAPP_NAME>
    sshPublicKey: <SSH_PUBLIC_KEY>
    provider:
      server: <API_URL>
      username: <USER_NAME>
      password: <PASSWORD>
      insecure: false
```

Где:

- `mainNetwork` — имя сети, в которой будут размещаться облачные узлы в VCD;
- `organization` — имя Organization в VCD;
- `virtualDataCenter` — имя Virtual Data Center в VCD;
- `virtualApplicationName` — имя vApp, где будут создаваться узлы, например `dkp-vcd-app`;
- `sshPublicKey` — публичный SSH-ключ для доступа к узлам;
- `provider.server` — URL-адрес API VCD;
- `provider.username` — имя пользователя VCD;
- `provider.password` — пароль пользователя VCD;

- `provider.insecure` — установите значение `true`, если VCD использует самоподписанный TLS-сертификат.

Если для аутентификации используется токен, вместо `username` и `password` укажите `apiToken`:

```
provider:
  server: <API_URL>
  apiToken: <API_TOKEN>
  insecure: false
```

2. Примените ModuleConfig:

```
d8 k apply -f cloud-provider-vcd-mc.yaml
d8 k get mc cloud-provider-vcd
```

3. Убедитесь, что все поды в неймспейсе `d8-cloud-provider-vcd` находятся в состоянии `Running`:

```
d8 k get pods -n d8-cloud-provider-vcd
```

4. Убедитесь, что в кластере созданы StorageClass для VCD:

```
d8 k get sc
```

5. Создайте файл с ресурсами VCDInstanceClass и NodeGroup. Например, `vcd-instanceclass-nodegroup.yaml`:

```
---
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: worker
spec:
  rootDiskSizeGb: 50
  sizingPolicy: <SIZING_POLICY>
  storageProfile: <STORAGE_PROFILE>
  template: <VAPP_TEMPLATE>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VCDInstanceClass
      name: worker
    maxPerZone: 2
    minPerZone: 1
  nodeTemplate:
    labels:
      node-role/worker: ""
```

6. Примените манифест:

```
d8 k apply -f vcd-instanceclass-nodegroup.yaml
```

После применения манифеста DKP начнёт создавать виртуальные машины в VCD, управляемые модулем `node-manager`.

7. Убедитесь, что в кластере появилось требуемое количество узлов:

```
d8 k get nodes -O wide
```

8. При сбоях создания VM проверьте объекты Machine, MachineSet и логи machine-controller-manager:

```
d8 k -n d8-cloud-instance-manager get machinesets,machines -O wide
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Добавление вручную созданных узлов через bootstrap-скрипт

Перед началом убедитесь, что выполнены следующие условия:

- Модуль `cloud-provider-vcd` включён:

```
d8 k get module cloud-provider-vcd -O wide
```

- Компоненты модуля `cloud-provider-vcd` находятся в состоянии `Running`:

```
d8 k -n d8-cloud-provider-vcd get pods -O wide
```

- В кластере созданы StorageClass для VCD:

```
d8 k get sc
```

- В VCD создана виртуальная машина, которая будет подключена к кластеру.
- Имя виртуальной машины в VCD совпадает с именем хоста (hostname) внутри операционной системы.
- В дополнительных параметрах VM в VCD задано значение:

```
disk.EnableUUID = TRUE
```

- Виртуальная машина подключена к сети VCD, используемой как основная сеть для облачных узлов кластера. Обычно это сеть, указанная в параметре `mainNetwork` конфигурации `cloud-provider-vcd` или в используемом `VCDInstanceClass`.
- На виртуальной машине установлен один из пакетных менеджеров (`apt`/`apt-get`, `yum` или `rpm`) для поддерживаемой ОС. В РЕД ОС по умолчанию могут отсутствовать `yum` и `which`, поэтому их необходимо заранее установить.

- Создайте файл с ресурсом NodeGroup и типом узлов CloudStatic. Например, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

- Убедитесь, что NodeGroup создана и синхронизирована:

```
d8 k get nodegroup cloud-static
```

Пример ожидаемого результата:

NAME	TYPE	READY	NODES	UPTODATE	INSTANCES	DESIRED	MIN	MAX
STANDBY	STATUS	AGE	SYNCED					
cloud-static	CloudStatic	0	0	0				
1m	True							

- Получите bootstrap-скрипт для созданной NodeGroup:

```
NODE_GROUP=cloud-static
```

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
-o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

- Скопируйте bootstrap-скрипт на подключаемую виртуальную машину:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

- Подключитесь к виртуальной машине по SSH:

```
ssh <USER>@<NODE_IP>
```

- На виртуальной машине назначьте права и запустите bootstrap-скрипт:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

После запуска bootstrap-скрипт установит необходимые компоненты, настроит container runtime, kubelet и подключит узел к кластеру.

- На master-узле проверьте появление нового узла:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
static-master-0	Ready	master	1h	v1.33.10	192.168.240.138
cloud-static-worker-0	Ready	cloud-static	5m	v1.33.10	192.168.240.151

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `cloud-provider-vcd`
- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `node-manager`

Гибридный кластер с vSphere

Далее описан процесс добавления узлов из vSphere в существующий статический кластер Deckhouse Kubernetes Platform (DKP).

Для интеграции с vSphere используется модуль `cloud-provider-vsphere`. Он обеспечивает взаимодействие DKP с vCenter, получение информации о виртуальных машинах, работу с параметрами размещения и интеграцию с инфраструктурными возможностями vSphere.

В разделе описаны два способа добавления узлов:

- **Автоматическое создание узлов в vSphere.** DKP создаёт виртуальные машины через API vSphere. Параметры ВМ задаются ресурсом `VsphereInstanceClass`, а требуемое количество узлов и зоны размещения — ресурсом `NodeGroup` с типом `CloudEphemeral`.
- **Подключение вручную созданных узлов через bootstrap-скрипт.** Виртуальная машина создаётся пользователем заранее и подключается к кластеру с помощью bootstrap-скрипта DKP. Для такого сценария используется `NodeGroup` с типом `CloudStatic`.

Предварительные требования для vSphere

Перед началом убедитесь, что выполнены следующие условия:

- Кластер создан с параметром `clusterType: Static`.
- Между сетью статических узлов и сетью виртуальных машин во vSphere настроена сетевая связность. Узлы vSphere, добавляемые в кластер, имеют доступ к Kubernetes API, DNS и необходимым адресам согласно разделам «Сетевое взаимодействие» и «Настройка сетевых политик». При использовании Cilium с туннелированием трафика подов выбран режим `tunnelMode`, соответствующий сетевой связности между площадками.
- Выполнены требования из раздела «Подключение и авторизация в VMware vSphere»:
 - настроен доступ к vCenter;
 - подготовлена учётная запись vSphere с необходимыми привилегиями;
 - подготовлен шаблон виртуальной машины;
 - настроены сети, Datastore, теги регионов и зон.

Добавление автоматически создаваемых узлов

Для подключения уже работающего статического кластера к vCenter используйте ресурс `ModuleConfig` модуля `cloud-provider-vsphere`.

В параметре `spec.settings` укажите параметры доступа к vCenter, сетевые настройки, теги региона и зоны, а также SSH-ключи, которые будут добавлены на создаваемые виртуальные машины.

Пример конфигурации и описание доступных параметров приведены в примерах модуля и в разделе с описанием настроек модуля.

1. Создайте файл, например `vsphere-mc.yaml`, с `ModuleConfig` для модуля `cloud-provider-vsphere`:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: "<VCENTER_FQDN>"
    username: "<USERNAME@DOMAIN.LOCAL>"
    password: "<PASSWORD>"
    insecure: true
    vmFolderPath: "<FOLDER_PATH_UNDER_DATACENTER>"
    regionTagCategory: "<TAG_CATEGORY_FOR_REGION>"
    zoneTagCategory: "<TAG_CATEGORY_FOR_ZONE>"
    region: "<REGION_TAG_NAME_ON_DATACENTER>"
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
    internalNetworkNames:
      - "<PORT_GROUP_NAME_FOR_INTERNAL_IP>"
    sshKeys:
      - "<SSH_PUBLIC_KEY_ONE_LINE>"

```

Значения параметров:

- `host` — адрес vCenter;
- `username`, `password` — учётные данные пользователя vSphere;
- `insecure` — отключение проверки TLS-сертификата vCenter;
- `vmFolderPath` — папка, в которой будут создаваться виртуальные машины;
- `regionTagCategory`, `zoneTagCategory` — категории тегов региона и зоны;
- `region` — тег региона;
- `zones` — список зон, в которых можно создавать узлы;
- `internalNetworkNames` — список сетей vSphere для подключения создаваемых узлов;
- `sshKeys` — публичные SSH-ключи, которые будут добавлены на создаваемые виртуальные машины.

2. Примените конфигурацию модуля:

```
d8 k apply -f vsphere-мс.yaml
```

3. Дождитесь готовности модуля `cloud-provider-vsphere`:

```
d8 k get module cloud-provider-vsphere -O wide
```

4. Создайте файл с ресурсами `VsphereInstanceClass` и `NodeGroup` со значением `nodeType: CloudEphemeral`. Например, `vsphere-instance.yaml`:

```

apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: ephemeral
spec:
  numCPUs: 2
  memory: 4096
  rootDiskSize: 40
  template: "<PATH_TO_TEMPLATE_FROM_DATACENTER>"
  mainNetwork: "<PORT_GROUP_NAME>"
  datastore: "<DATASTORE_OR_FOLDER/DATASTORE>"
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ephemeral
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VsphereInstanceClass
      name: ephemeral
    maxPerZone: 1
    minPerZone: 1
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
  disruptions:
    approvalMode: Automatic

```

Где:

- `VsphereInstanceClass` описывает параметры виртуальной машины, которая будет создана во vSphere;
- `NodeGroup` описывает группу узлов, которую DKP должен поддерживать в кластере;
- `nodeType: CloudEphemeral` означает, что узлы будут создаваться автоматически через облачный провайдер;
- `cloudInstances.classReference` указывает на `VsphereInstanceClass`;
- `cloudInstances.zones` должен содержать зоны из списка `zones` в `ModuleConfig`.

5. Примените манифест:

```
d8 k apply -f vsphere-instance.yaml
```

После применения манифеста DKP начнёт создавать виртуальную машину во vSphere. После загрузки ВМ kubelet подключится к Kubernetes API, и новый узел появится в кластере.

6. Проверьте состояние узлов:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION
static-master-0	Ready	control-plane,master	1h	v1.33.10
ephemeral-1ca02a5b-7588b-k89dc	Ready	ephemeral	10m	v1.33.10

7. При сбоях создания ВМ проверьте объекты Machine, MachineSet и логи machine-controller-manager:

```

d8 k -n d8-cloud-instance-manager get machinesets,machines -O wide
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200

```

Также проверьте события в кластере:

```
d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
```

Добавление вручную созданных узлов через bootstrap-скрипт

Перед началом убедитесь, что выполнены следующие условия:

- Модуль `cloud-provider-vsphere` включён:

```
d8 k get module cloud-provider-vsphere -O wide
```

- Компоненты модуля `cloud-provider-vsphere` находятся в состоянии `Running`:

```
d8 k -n d8-cloud-provider-vsphere get pods -O wide
```

- В кластере созданы StorageClass для vSphere:

```
d8 k get sc
```

- В vSphere создана виртуальная машина, которая будет подключена к кластеру.
- Имя виртуальной машины в vSphere совпадает с именем хоста (hostname) внутри операционной системы.
- В дополнительных параметрах VM в vSphere заданы параметры:

```
disk.EnableUUID = TRUE
guestinfo.metadata = <BASE64_ENCODED_METADATA>
guestinfo.metadata.encoding = base64
```

Параметр `guestinfo.metadata` должен содержать конфигурацию метаданных, закодированных в Base64. Пример файла `metadata.json`:

```
{
  "instance-id": "cloud-static-worker-0",
  "local-hostname": "cloud-static-worker-0",
  "public-keys-data": "<SSH_PUBLIC_KEY>",
  "network": {
    "version": 2,
    "ethernets": {
      "id0": {
        "match": {
          "driver": "vmxnet3"
        },
        "set-name": "ens192",
        "dhcp4": true
      }
    }
  }
}
```

Где:

- `instance-id` — идентификатор виртуальной машины;
- `local-hostname` — имя хоста (hostname) узла внутри операционной системы;
- `public-keys-data` — публичный SSH-ключ для доступа к виртуальной машине;
- `network` — сетевые настройки, которые будут применены внутри виртуальной машины.

Чтобы получить значение для параметра `guestinfo.metadata`, выполните:

```
METADATA_B64="$(base64 -w0 metadata.json)"
echo "$METADATA_B64"
```

- Виртуальная машина подключена к сети, указанной в параметре `internalNetworkNames` конфигурации модуля `cloud-provider-vsphere`.
- На виртуальной машине установлен один из пакетных менеджеров (`apt`/`apt-get`, `yum` или `rpm`) для поддерживаемой ОС. В РЕД ОС по умолчанию могут отсутствовать `yum` и `which`, поэтому их необходимо заранее установить.

- Создайте файл с ресурсом NodeGroup и типом узлов `CloudStatic`. Например, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

- Убедитесь, что NodeGroup создана и синхронизирована:

```
d8 k get nodegroup cloud-static
```

Пример ожидаемого результата:

NAME	TYPE	READY	NODES	UPTODATE	INSTANCES	DESIRED	MIN	MAX
STANDBY	STATUS	AGE	SYNCED					
cloud-static	CloudStatic	0	0	0				
1m	True							

- Получите bootstrap-скрипт для созданной NodeGroup:

```
NODE_GROUP=cloud-static
```

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
-o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

4. Скопируйте bootstrap-скрипт на подключаемую виртуальную машину:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

5. Подключитесь к виртуальной машине по SSH:

```
ssh <USER>@<NODE_IP>
```

6. На виртуальной машине назначьте права и запустите bootstrap-скрипт:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

После запуска bootstrap-скрипт установит необходимые компоненты, настроит container runtime, kubelet и подключит узел к кластеру.

7. На master-узле проверьте появление нового узла:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
static-master-0	Ready	master	1h	v1.33.10	192.168.240.135
cloud-static-worker-0	Ready	cloud-static	5m	v1.33.10	192.168.240.152

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `cloud-provider-vsphere`
- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `node-manager`

Гибридный кластер с DVP

Далее описан процесс добавления узлов из Deckhouse Virtualization Platform (DVP) в существующий статический кластер DKP.

Для интеграции с DVP используется модуль `cloud-provider-dvp`. Он обеспечивает взаимодействие DKP с API кластера DVP, создание виртуальных машин, подключение созданных ВМ к существующему Kubernetes-кластеру и управление жизненным циклом узлов через механизмы Cluster API.

В разделе описаны два способа добавления узлов:

- **Автоматическое создание узлов в DVP.** DKP создаёт виртуальные машины через API DVP. Параметры ВМ задаются ресурсом `DVPInstanceClass`, а требуемое количество узлов и зоны размещения — ресурсом `NodeGroup` с типом `CloudEphemeral`.
- **Подключение вручную созданных узлов через bootstrap-скрипт.** Виртуальная машина создаётся пользователем заранее и подключается к кластеру с помощью bootstrap-скрипта DKP. Для такого сценария используется `NodeGroup` с типом `CloudStatic`.

Предварительные требования

Перед началом убедитесь, что выполнены следующие условия:

- Кластер создан с параметром `clusterType: Static`.
- Между сетью статических узлов кластера DKP и сетью виртуальных машин в DVP настроена сетевая связность. Создаваемые в DVP узлы имеют доступ к Kubernetes API подключаемого DKP-кластера, DNS и необходимым адресам согласно разделам Сетевое взаимодействие и «Настройка сетевых политик». При использовании Cilium с туннелированием трафика подов выбран режим `tunnelMode`, соответствующий сетевой связности между площадками. Из кластера DKP доступен Kubernetes API кластера DVP.
- Выполнены требования из раздела «Подготовка окружения»:
 - создан `ServiceAccount` для доступа к API DVP;
 - сгенерирован `kubeconfig` для подключения к API DVP;
 - подготовлен неймспейс, в котором будут создаваться виртуальные машины и диски.
- В DVP доступен образ ОС Linux с поддержкой `cloud-init`, например `ubuntu-24-04-lts`.
- В DVP доступен подходящий `VirtualMachineClass`, например `amd-epyc-gen-3`.
- В DVP доступен `StorageClass` для корневых дисков виртуальных машин, например `replicated`.
- Если используется шаблон виртуальной машины, убедитесь, что он содержит только один диск.

⚠ В параметрах `DVPInstanceClass` используются ресурсы кластера DVP: `VirtualMachineClass`, `ClusterVirtualImage`, `VirtualImage`, `VirtualDisk` и `StorageClass` из DVP, а не из подключаемого DKP-кластера.

⚠ Убедитесь, что в статическом DKP-кластере отсутствуют `StorageClass` с именами, совпадающими с именами `StorageClass` в кластере DVP.

При включении модуля `cloud-provider-dvp` соответствующие `StorageClass` автоматически синхронизируются из DVP в DKP-кластер. Если `StorageClass` с таким именем уже существует в статическом кластере, может возникнуть конфликт ресурсов, из-за которого установка или обновление модуля завершится с ошибкой.

Добавление автоматически создаваемых узлов

1. На машине администратора, где настроен доступ к кластеру DVP, подготовьте `kubeconfig` для доступа модуля `cloud-provider-dvp` к API DVP.

Выполните шаги из раздела «Подготовка окружения» и закодируйте полученный `kubeconfig` в Base64:

```
export DVP_PROVIDER_KUBECONFIG="./kubeconfig"
export DVP_KUBECONFIG_B64="$(base64 -w0 ${DVP_PROVIDER_KUBECONFIG})"
```

2. Задайте неймспейс DVP, в котором будут создаваться виртуальные машины и диски:

```
export DVP_NAMESPACE="<DVP_NAMESPACE>"
```

3. Укажите зону DVP, в которой будут создаваться узлы.

На данный момент зонирование в DVP находится в разработке, поэтому для параметров `zones` в `ModuleConfig` и `NodeGroup` используйте значение `default`:

```
export DVP_ZONE="default"
```

При необходимости можно проверить топологические метки узлов в кластере DVP:

```
d8 k get nodes -L topology.kubernetes.io/region,topology.kubernetes.io/zone
```

⚠ Значение зоны в `ModuleConfig` и `NodeGroup` должно совпадать. Пока в DVP доступно только значение `default`.

4. Создайте файл, например `cloud-provider-dvp-mc.yaml`, с конфигурацией модуля `cloud-provider-dvp`:

```
cat > cloud-provider-dvp-mc.yaml <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  enabled: true
  version: 1
  settings:
    provider:
      kubeconfigDataBase64: ${DVP_KUBECONFIG_B64}
      namespace: ${DVP_NAMESPACE}
    zones:
      - ${DVP_ZONE}
EOF
```

В манифесте используются значения переменных окружения, заданных на предыдущих шагах: `DVP_KUBECONFIG_B64`, `DVP_NAMESPACE` и `DVP_ZONE`.

5. Примените `ModuleConfig`:

```
d8 k apply -f cloud-provider-dvp-mc.yaml
```

6. Дождитесь включения модуля `cloud-provider-dvp`:

```
d8 k get module cloud-provider-dvp -O wide
```

Модуль должен перейти в состояние `Ready`, а поды в неймспейсе `d8-cloud-provider-dvp` — в состояние `Running`.

7. Убедитесь, что модуль `node-manager` находится в состоянии `Ready`:

```
d8 k get module node-manager -O wide
```

Если модуль находится в состоянии `Error`, проверьте, что в `ModuleConfig` и `NodeGroup` указаны доступные зоны DVP.

8. Убедитесь, что в кластере появился ресурс `DVPInstanceClass`:

```
d8 k get crd dvpinstanceclasses.deckhouse.io
```

9. На машине администратора, где настроен доступ к кластеру DVP, проверьте доступные классы виртуальных машин, образы и `StorageClass`:

```
d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get virtualmachineclasses
d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get clustervirtualimages
d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get storageclasses
```

Используйте полученные значения при создании `DVPInstanceClass`.

10. Создайте файл, например `dvp-instanceclass-nodegroup.yaml`, с ресурсами `DVPInstanceClass` и `NodeGroup`:

```
apiVersion: deckhouse.io/v1alpha1
kind: DVPInstanceClass
metadata:
  name: dvp-worker
spec:
  virtualMachine:
    cpu:
      cores: 3
      coreFraction: 20%
    memory:
      size: 6Gi
    virtualMachineClassName: <VIRTUAL_MACHINE_CLASS_NAME>
    bootloader: EFI
  rootDisk:
    size: 15Gi
    storageClass: <STORAGE_CLASS_NAME>
  image:
    kind: ClusterVirtualImage
    name: <CLUSTER_VIRTUAL_IMAGE_NAME>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: dvp-worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: DVPInstanceClass
      name: dvp-worker
    minPerZone: 1
    maxPerZone: 1
    zones:
      - <ZONE_NAME>
```

Где:

- `virtualMachineClassName` — имя `VirtualMachineClass` в DVP, например `amd-е-пус-gen-3`;
- `rootDisk.storageClass` — имя `StorageClass` в DVP, например `replicated`;
- `rootDisk.image.kind` — тип источника образа. Для кластерного образа используйте `ClusterVirtualImage`;
- `rootDisk.image.name` — имя образа ОС в DVP, например `ubuntu-24-04-lts`;
- `cloudInstances.zones` — зона DVP, в которой будет создан узел. Значение должно входить в список `zones` из `ModuleConfig`.

11. Примените манифест:

```
d8 k apply -f dvp-instanceclass-nodegroup.yaml
```

После применения DKP начнёт создавать виртуальную машину в DVP и подключать её к кластеру как узел.

12. Проверьте состояние `NodeGroup`:

```
d8 k get nodegroup dvp-worker -O wide
d8 k describe nodegroup dvp-worker
```

13. Проверьте появление нового узла в DKP-кластере:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
dvp-hybrid-master-0	Ready	control-plane,master	1h	v1.33.10	10.12.0.69
dvp-worker-c75a75c1-twqp4-bjpv1	Ready	dvp-worker	10m	v1.33.10	10.12.3.15

Добавление вручную созданных узлов через bootstrap-скрипт

Перед началом убедитесь, что выполнены следующие условия:

- Модуль `cloud-provider-dvp` включён:

```
d8 k get module cloud-provider-dvp -O wide
```

- Компоненты модуля `cloud-provider-dvp` находятся в состоянии `Running`:

```
d8 k -n d8-cloud-provider-dvp get pods -O wide
```

- В DVP создана виртуальная машина, которая будет подключена к кластеру.
- Виртуальная машина подключена к сети DVP, используемой для гибридной интеграции с кластером.
- IP-адрес виртуальной машины входит в диапазон, указанный в `internalNetworkCIDsRs`.
- Имя виртуальной машины в DVP совпадает с `hostname` внутри операционной системы.
- На виртуальной машине есть SSH-доступ для копирования и запуска `bootstrap`-скрипта.
- Пользователь для подключения по SSH может выполнять команды через `sudo` без ввода пароля.
- На виртуальной машине установлен один из пакетных менеджеров (`apt`/`apt-get`, `yum` или `rpm`) для поддерживаемой ОС. В РЕД ОС по умолчанию могут отсутствовать `yum` и `which`, поэтому их необходимо заранее установить.

- Создайте файл, например `cloud-static-nodegroup.yaml`, с ресурсом `NodeGroup` и типом узлов `CloudStatic`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

- Примените манифест:

```
d8 k apply -f cloud-static-nodegroup.yaml
```

- Убедитесь, что `NodeGroup` создана и синхронизирована:

```
d8 k get nodegroup cloud-static
```

Пример ожидаемого результата:

NAME	TYPE	READY	NODES	UPTODATE	INSTANCES	DESIRED	MIN	MAX
STANDBY	STATUS	AGE	SYNCED					
cloud-static	CloudStatic	0	0	0				
1m	True							

- Получите `bootstrap`-скрипт для созданной `NodeGroup`:

```
NODE_GROUP=cloud-static
```

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
-o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

- Скопируйте `bootstrap`-скрипт на подключаемую виртуальную машину:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

- Подключитесь к виртуальной машине по SSH:

```
ssh <USER>@<NODE_IP>
```

7. На виртуальной машине декодируйте bootstrap-скрипт, назначьте права и запустите его:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

После запуска bootstrap-скрипт установит необходимые компоненты, настроит container runtime, kubelet и подключит узел к кластеру.

8. На master-узле проверьте появление нового узла:

```
d8 k get nodes -O wide
```

Пример ожидаемого результата:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
dvp-hybrid-master-0	Ready	control-plane,master	1h	v1.33.12	10.12.0.69
cloud-static-worker-0	Ready	cloud-static	5m	v1.33.12	10.12.3.88

9. При сбоях подключения проверьте состояние NodeGroup, события и логи bootstrap на подключаемой виртуальной машине:

```
d8 k get nodegroup cloud-static
d8 k describe nodegroup cloud-static
d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
```

На подключаемой виртуальной машине:

```
sudo tail -n 120 /var/log/d8/bashible/bootstrap.log
```

Если в логах есть ошибка `Failed to discover node ip that matches internalNetworkCIDRs`, проверьте, что IP-адрес виртуальной машины входит в `internalNetworkCIDRs`.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `cloud-provider-dvp`
- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `node-manager`
- Документация модуля `virtualization`

Высокая надежность и доступность

Надёжность и устойчивость Kubernetes-кластера — ключевые характеристики, определяющие стабильность инфраструктуры. Deckhouse Kubernetes Platform (DKP) обеспечивает высокую доступность (High Availability, HA) и отказоустойчивость за счёт встроенных механизмов и модулей.

При включенном режиме HA критически важные компоненты кластера запускаются с учетом нужной избыточности что гарантирует их непрерывную работу. Даже при отказе отдельного экземпляра сервисы могут функционировать без простоев. Подробнее о включении режима HA – в разделе Управление режимом HA.

Если в кластере больше одного master-узла, режим HA включается автоматически — как при первичном развёртывании, так и при последующем добавлении узлов. Рекомендуемые роли и количество узлов приведены в разделе Рекомендации по конфигурации узлов кластера и предотвращению перегрузки.

Для проверки устойчивости в DKP предусмотрены инструменты хаос-инжиниринга. С их помощью можно управлять или в случайном порядке выводить из строя отдельные компоненты и отслеживать реакцию инфраструктуры. О настройке инструментов хаос-инжиниринга читайте в разделе Хаос-инжиниринг.

Повышенную отказоустойчивость кластера можно реализовать через межкластерное взаимодействие на базе режима Service Mesh модуля `istio`. В этом режиме настраивается федерация между несколькими кластерами: при сбоях в одном из них нагрузка автоматически перераспределяется на другие. Подробнее про настройку читайте в разделе Федерация.

Дополнительные ресурсы

- Документация модуля `istio`

Рекомендации по конфигурации узлов кластера и предотвращению перегрузки

Правильная конфигурация узлов повышает отказоустойчивость кластера и упрощает его сопровождение. Ниже описаны рекомендуемые роли и количество узлов для различных компонентов.

Master-узлы

Для обеспечения отказоустойчивости кластера **всегда** используйте не менее трёх master-узлов. Такое количество обеспечит безотказную работу кластера и позволит безопасно обновлять master-узлы. В большем числе нет

необходимости, а двух узлов будет недостаточно для обеспечения согласия между master-узлами (кворума) в случае возникновения неполадок с одним из них.

При использовании всего одного master-узла его отказ приведет к сбою всего кластера, так как именно master-узел управляет ключевыми компонентами кластера, обеспечивающими его работу.

Frontend-узлы

Frontend-узлы балансируют входящий трафик, на них работают Ingress-контроллеры. Используйте более одного frontend-узла. Frontend-узлы должны выдерживать трафик при отказе как минимум одного frontend-узла.

Например, если в кластере два frontend-узла, то каждый из них должен справляться со всей нагрузкой на кластер в случае, если второй выйдет из строя. Если в кластере три frontend-узла, то каждый должен выдерживать увеличение нагрузки как минимум в полтора раза.

Узлы мониторинга

Узлы мониторинга служат для запуска Grafana, Prometheus и других компонентов мониторинга. В нагруженных кластерах со множеством алертов и большими объемами метрик под мониторинг рекомендуется выделить отдельные узлы. Если этого не сделать, компоненты мониторинга будут размещены на системных узлах.

При выделении узлов под мониторинг важно, чтобы на них были быстрые диски (не менее 400 IOPS).

Системные узлы

Системные узлы предназначены для запуска модулей Deckhouse. Выделите два системных узла: в этом случае модули Deckhouse будут работать на них, не пересекаясь с пользовательскими приложениями кластера.

Предотвращение перегрузки Kubernetes API (FlowSchema)

В кластере под управлением DKP по умолчанию включен компонент, реализующий FlowSchema и PriorityLevelConfiguration для предотвращения перегрузки Kubernetes API.

FlowSchema устанавливает PriorityLevel для list-запросов от всех сервис-аккаунтов в пространствах имен Deckhouse (у которых установлен label `heritage: deckhouse`) к следующим apiGroup:

- `v1` (Pod, Secret, ConfigMap, Node и т. д.) — в кластерах с большим количеством основных ресурсов (например, секретов или подов).
- `apps/v1` (DaemonSet, Deployment, StatefulSet, ReplicaSet и т. д.) — при развертывании большого количества приложений в кластере (например, Deployment'ов).
- `deckhouse.io` (кастомные ресурсы Deckhouse) — при наличии большого числа различных кастомных ресурсов Deckhouse в кластере.
- `cilium.io` (кастомные ресурсы Cilium) — в кластерах с большим количеством политик Cilium.

Все запросы к API, соответствующие FlowSchema, помещаются в одну очередь обработки с общим приоритетом. Это ограничивает нагрузку на Kubernetes API и предотвращает его перегрузку при массовом выполнении list-запросов.

Компонент не имеет настроек, но доступны следующие команды:

- Проверка состояния уровней приоритета (priority level):

```
d8 k get --raw /debug/api_priority_and_fairness/dump_priority_levels
```

- Проверка состояния очередей уровней приоритета:

```
d8 k get --raw /debug/api_priority_and_fairness/dump_queues
```

Также он передаёт в Grafana следующие метрики:

- `apiserver_flowcontrol_rejected_requests_total` — общее число отброшенных запросов;
- `apiserver_flowcontrol_dispatched_requests_total` — общее число обработанных запросов;
- `apiserver_flowcontrol_current_inqueue_requests` — количество запросов в очередях;
- `apiserver_flowcontrol_current_executing_requests` — количество запросов в обработке.

Задержка перезагрузки или выключения узла, пока на нём работают критичные поды

i

Доступно в редакции **EE**.

В DKP поддерживается механизм включения задержки выключения или перезагрузки узла (далее просто «выключение узла»), если на нем запущены stateful-приложения или виртуальные машины (ВМ). Механизм реализуется сервисом `d8-shutdown-inhibitor` модуля `node-manager`.

Задержка срабатывает, если на узле запущены поды с лейблом `pod.deckhouse.io/inhibit-node-shutdown`. DKP откладывает выключение узла, чтобы пользователь смог перенести критичные поды на другой узел или завершить их работу штатным образом. Это полезно, например, для stateful-нагрузок и других случаев, когда необходимо избежать потери данных. Подробная информация о принципах работы и особенностях механизма представлена в документации модуля `node-manager`.

Включение задержки выключения узла



Для принятия решения о блокировке выключения DKP дополнительно опрашивает NodeGroup. Если текущий узел принадлежит к группе `master` и является единственным master-узлом в кластере, блокировка выключения для него применена не будет.

Чтобы включить механизм задержки выключения узла, добавьте на под лейбл `pod.deckhouse.io/inhibit-node-shutdown` (для Deployment укажите лейбл в шаблоне пода).

Пример манифеста пода с лейблом `pod.deckhouse.io/inhibit-node-shutdown`:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```



Блокировка срабатывает только для подов с лейблом `pod.deckhouse.io/inhibit-node-shutdown` в фазе `Running` на узле. Поды без лейбла kubelet выключает по обычному сценарию graceful shutdown.

Поиск подов, блокирующих выключение, и диагностика на узле

Для поиска подов, из-за которых блокируется выключение узла, используйте команду:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -O wide
```

Для просмотра состояния `GracefulShutdownPostpone` на узле используйте команду:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?(@.type=="GracefulShutdownPostpone")]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}}{end}'
```

Возможные значения поля `reason`:

- `WaitingForShutdownSignal` — на узле работает механизм блокировки выключения, но блокировка ещё не началась;
- `PodsWithLabelAreRunningOnNode` — выключение заблокировано, поскольку на узле ещё работают поды с лейблом `pod.deckhouse.io/inhibit-node-shutdown`;
- `NoRunningPodsWithLabel` — блокировка снята, выключение может быть продолжено.

Дополнительные ресурсы

- Документация модуля `node-manager`

Управление режимом HA



Обратите внимание, что если в кластере **более одного master-узла**, режим HA **включается автоматически**. Это справедливо как для развёртывания кластера сразу с тремя master-узлами, так и при увеличении количества master-узлов с одного до трёх.

Глобальное включение режима HA

Включить режим отказоустойчивости для всей платформы DKP можно одним из следующих способов.

Через кастомный ресурс `ModuleConfig/global`

1. Установите в `ModuleConfig/global` параметр `settings.highAvailability` в значение `true`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings:
    highAvailability: true
```

- Убедитесь, что режим включился. Например, проверьте количество подов `deckhouse` в пространстве имён `d8-system`, выполнив команду:

```
d8 k -n d8-system get po | grep deckhouse
```

Количество подов `deckhouse` должно быть больше одного, как показано в примере вывода ниже:

deckhouse-57695f4d68-8rk6l	2/2	Running	0	3m49s
deckhouse-5764gfud68-76dsb	2/2	Running	0	3m49s
deckhouse-fgrhy4536s-fhu6s	2/2	Running	0	3m49s

Через веб-интерфейс Deckhouse

Если в кластере включен модуль `console`, откройте веб-интерфейс Deckhouse, перейдите в раздел «Deckhouse» — «Глобальные настройки» — «Глобальные настройки модулей» и установите переключатель «Режим отказоустойчивости» в положение «Да».

Настройка режима HA с двумя master-узлами и arbiter-узлом

В Deckhouse Kubernetes Platform возможна настройка режима HA с двумя master-узлами и arbiter-узлом. Такой подход позволяет обеспечить требования по HA в условиях ограниченных ресурсов.

На arbiter-узле размещается только etcd, без остальных компонентов control plane. Этот узел используется для обеспечения кворума etcd.

Требования к arbiter-узлу:

- не менее 2 ядер CPU;
- не менее 4 ГБ RAM;
- не менее 8 ГБ дискового пространства под etcd.

Требования к сетевым задержкам для arbiter-узла аналогичны требованиям для master-узлов.

Настройка в облачном кластере

Пример ниже актуален для облачного кластера с тремя master-узлами. Чтобы настроить режим HA с двумя master-узлами и arbiter-узлом в облачном кластере, необходимо удалить из кластера один master-узел и добавить один arbiter-узел.

Для этого выполните следующие действия:

⚠ Описанные ниже шаги необходимо выполнять с первого по порядку master-узла кластера (`master-0`). Это связано с тем, что кластер всегда масштабируется по порядку: например, невозможно удалить узлы `master-0` и `master-1`, оставив `master-2`.

⚠ Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений рекомендуется создать резервную копию данных модуля.

- Сделайте резервную копию etcd и директории `/etc/kubernetes`.
- Скопируйте полученный архив за пределы кластера (например, на локальную машину).
- Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
- Убедитесь, что очередь DKP пуста:

```
d8 system queue list
```

- На локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес container registry при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

- В контейнере с инсталлятором** выполните следующую команду:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
--ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

Измените настройки облачного провайдера:

- В параметре `masterNodeGroup.replicas` укажите 2.
- Создайте NodeGroup для arbiter-узла. На arbiter-узле **обязательно** должен быть лейбл

`node.deckhouse.io/etcd-arbiter: ""` и taint, предотвращающий размещение на нем пользовательской нагрузки. Пример описания NodeGroup для arbiter-узла:

```
nodeGroups:
- name: arbiter
  replicas: 1
  nodeTemplate:
    labels:
      node.deckhouse.io/etcd-arbiter: ""
    taints:
      - key: node.deckhouse.io/etcd-arbiter
        effect: NoSchedule
  zones:
    - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
# ... остальная часть манифеста
```

- Сохраните изменения.

i

Для **Yandex Cloud** при использовании внешних адресов на master-узлах количество элементов массива в параметре `masterNodeGroup.instanceClass.externalIPAddresses` должно равняться количеству master-узлов. При использовании значения `Auto` (автоматический заказ публичных IP-адресов) количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при одном master-узле (`masterNodeGroup.replicas: 1`) и автоматическом заказе адресов параметр `masterNodeGroup.instanceClass.externalIPAddresses` будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
```

- В контейнере с инсталлятором выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST>
```

i

Важно. Для **OpenStack** и **VK Cloud (OpenStack)** после подтверждения удаления узла обязательно проверьте удаление диска `<prefix>kubernetes-data-N` в самом OpenStack.

Например, при удалении узла `cloud-demo-master-2` в веб-интерфейсе OpenStack или в OpenStack CLI необходимо проверить отсутствие диска `cloud-demo-kubernetes-data-2`.

В случае, если диск `kubernetes-data` останется, при увеличении количества master-узлов могут возникнуть проблемы в работе etcd.

- Проверьте очередь Deckhouse с помощью следующей команды и убедитесь, что отсутствуют ошибки:

```
d8 system queue list
```

Настройка в статическом кластере

Чтобы настроить режим HA с двумя master-узлами и arbiter-узлом в статическом кластере, выполните следующие действия:

- Создайте NodeGroup для arbiter-узла. На arbiter-узле **обязательно** должен быть лейбл `node.deckhouse.io/etcd-arbiter: ""` и taint, предотвращающий размещение на нем пользовательской нагрузки. Пример описания NodeGroup для arbiter-узла:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: arbiter
spec:
  nodeType: Static
  nodeTemplate:
    labels:
      node.deckhouse.io/etcd-arbiter: ""
    taints:
      - key: node.deckhouse.io/etcd-arbiter
        effect: NoSchedule
# ... остальная часть манифеста
```

- Добавьте удобным вам способом в кластер узел, который будет использовать как arbiter-узел.

3. Убедитесь, что добавленный arbiter-узел находится в списке членов кластера etcd.
4. Удалите один master-узел из кластера.

Включение режима HA для отдельных компонентов

Некоторые модули DKP могут иметь собственные настройки режима HA. Чтобы включить режим высокой надежности в конкретном модуле, установите параметр `settings.highAvailability` в его настройках. При этом работа режима HA в отдельных модулях не зависит от состояния глобального режима HA.

Перечень модулей, для которых доступно управление режимом HA:

- `deckhouse`;
- `openvpn`;
- `istio`;
- `dashboard`;
- `multitenancy-manager`;
- `user-authn`;
- `ingress-nginx`;
- `prometheus-monitoring`;
- `monitoring-kubernetes`;
- `snapshot-controller`.

Чтобы вручную включить режим HA в конкретном модуле, добавьте в его конфигурацию параметр `settings.highAvailability`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    highAvailability: true
```

Чтобы убедиться, что режим включился, проверьте количество подов выбранного модуля. Например, для проверки работы режима в модуле `deckhouse`, проверьте количество подов в пространстве имён `d8-system`, выполнив следующую команду:

```
d8 k -n d8-system get po | grep deckhouse
```

Количество подов `deckhouse` должно быть больше одного:

<code>deckhouse-57695f4d68-8rk6l</code>	2/2	Running	0	3m49s
<code>deckhouse-5764gfud68-76dsb</code>	2/2	Running	0	3m49s
<code>deckhouse-fgrhy4536s-fhu6s</code>	2/2	Running	0	3m49s

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `console`
- Документация модуля `control-plane-manager`
- Документация модуля `deckhouse`
- Документация модуля `openvpn`
- Документация модуля `stronghold`

Хаос-инжиниринг



Режим хаос-инжиниринга включается только для групп узлов с `nodeType: CloudEphemeral`.

Включите режим хаос-инжиниринга для конкретных групп узлов (NodeGroup) одним из следующих способов:

1. Укажите в параметрах тестируемой группы узлов параметр `spec.chaos` с двумя вложенными параметрами:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

Здесь:

- `mode` — режим работы, доступны два варианта:
 - `DrainAndDelete` — при срабатывании выполняет drain узла, затем удаляет его.
 - `Disabled` — не воздействует на данную NodeGroup.
- `period` — интервал между срабатываниями Chaos Monkey. Задается в виде строки с указанием часов и минут: `30m`, `1h`, `2h30m`, `24h`.

Пример настройки для группы узлов:

```
# NodeGroup для облачных узлов в AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  chaos:
    mode: DrainAndDelete
    period: 24h
```

- Если в кластере включен модуль `console`, откройте веб-интерфейс Deckhouse, перейдите в настройки выбранной группы узлов в разделе «Узлы» — «Группы узлов» и включите Chaos Monkey в пункте «Параметры chaos monkey», указав временные интервалы в соответствующих полях.

Дополнительные ресурсы

- Документация модуля `console`
- Кастомные ресурсы модуля `node-manager`

Масштабирование платформы и управление узлами

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные механизмы для комплексного управления архитектурой кластера — как на уровне управляющего слоя (control plane), так и на уровне узлов.

Функции управления включают:

- Управление компонентами control plane — автоматизация настройки и обновления Kubernetes API, etcd и других системных компонентов; выпуск и продление сертификатов; масштабирование управляющего слоя и переход между режимами single-master и multi-master; отказоустойчивость и восстановление.
- Управление узлами — создание и масштабирование групп узлов с помощью NodeGroup; автоматическое развёртывание и обновление узлов; поддержка различных типов инфраструктуры (облако и bare-metal); управление статическими и облачными узлами; использование дополнительных инструментов для тонкой настройки.

Эти возможности позволяют создавать надёжные, масштабируемые и самовосстанавливающиеся кластеры, адаптируя их под любые требования производительности, отказоустойчивости и инфраструктурных ограничений.

В следующих разделах представлены подробные описания возможностей, примеры конфигураций и рекомендации по эффективному использованию управления control plane и узлами в Deckhouse Kubernetes Platform.

Общее управление и конфигурация control plane

Основные возможности

Deckhouse Kubernetes Platform (DKP) управляет компонентами управляющего слоя кластера (control plane) с помощью модуля `control-plane-manager`. Этот модуль запускается на всех управляющих узлах (master-узлах) с лейблом `node-role.kubernetes.io/control-plane: ""`.

Функционал управления control plane включает:

- Управление сертификатами необходимых для работы control plane, в том числе их продление и выпуск при изменении конфигурации. Автоматически поддерживается безопасная конфигурация и возможность быстрого добавления дополнительных SAN для организации защищённого доступа к API Kubernetes.
- Настройка компонентов. DKP генерирует все необходимые конфигурации и манифесты (kube-apiserver, etcd и др.), снижая вероятность ручных ошибок.
- Upgrade/downgrade компонентов. DKP поддерживает согласованное обновление или понижение версии control plane, что позволяет поддерживать единообразие версий в кластере. В редакциях CSE поддерживается только upgrade.
- Управление конфигурацией etcd-кластера и его членов. DKP масштабирует master-узлы, выполняет миграцию из single-master в multi-master и обратно.
- Настройка kubeconfig. DKP формирует актуальный конфигурационный файл (с правами `cluster-admin`), генерирует, продлевает и обновляет kubeconfig для компонентов control plane и административный kubeconfig (`admin.conf`), а также создает символическую ссылку для пользователя `root` (`/root/.kube/config -> admin.conf`). При включённом модуле `user-authz` символическую ссылку можно отключить параметром `rootKubeconfigSymlink` в модуле `control-plane-manager` (подробнее — в «FAQ» модуля `control-plane-manager`). Также DKP ужесточает права доступа к файлам `admin.conf` и `super-admin.conf` для усиления безопасности.



Некоторые параметры, влияющие на работу control plane, берутся из ресурса `ClusterConfiguration`.

Включение, отключение и настройка модуля

Включение / отключение

Включить или выключить модуль `control-plane-manager` можно следующими способами:

1. Создайте (или измените) ресурс `ModuleConfig/control-plane-manager`, указав `spec.enabled: true` или `false`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
```

2. Используйте команду:

```
d8 system module enable control-plane-manager
```

или

```
d8 system module disable control-plane-manager
```

3. Через веб-интерфейс Deckhouse:

- перейдите в раздел «Deckhouse — «Модули»;
- найдите модуль `control-plane-manager` и нажмите на него;
- включите тумблер «Модуль включен».

Настройка

Чтобы настроить модуль, используйте `ModuleConfig/control-plane-manager` и укажите необходимые параметры в `spec.settings`.

Пример с указанием версии схемы, включённым модулем и несколькими настройками:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
        - bakery.infra
        - devs.infra
      loadBalancer: {}
```

Проверка состояния и очередей DKP

Как проверить, что `control-plane-manager` корректно запущен и не находится в состоянии ожидания, а также как проверить активные задачи (очереди) в DKP:

1. Убедитесь, что модуль включён:

```
d8 k get modules control-plane-manager
```

2. Проверьте состояние подов `control-plane-manager` (поды находятся в пространстве имён `kube-system` и имеют лейбл `app=d8-control-plane-manager`):

```
d8 k -n kube-system get pods -l app=d8-control-plane-manager -o wide
```

Убедитесь, что все поды находятся в статусе `Running` (или `Completed`).

3. Проверьте, что master-узлы в состоянии `Ready`:

```
d8 k get nodes -l node-role.kubernetes.io/control-plane
```

Если требуется посмотреть более подробную информацию:

```
d8 k describe node <имя-узла>
```

4. Получите список очередей и активных заданий:

```
d8 system queue list
```


Пример вывода:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```



Перед выполнением «тяжёлых» процедур (например, перед переводом кластера из single-master в multi-master или перед обновлением версии Kubernetes) рекомендуется дождаться, чтобы все задачи в очередях были выполнены.

Управление сертификатами

В DKP за выпуск и продление всех SSL-сертификатов компонентов control plane отвечает модуль `control-plane-manager`. Он контролирует:

1. **Серверные сертификаты** для kube-apiserver и etcd, хранящиеся в секрете `d8-pki` (пространство имён `kube-system`):
 - Корневой CA Kubernetes (`ca.crt`, `ca.key`);
 - Корневой CA etcd (`etcd/ca.crt`, `etcd/ca.key`);
 - RSA-сертификат и ключ для подписи Service Account'ов (`sa.pub`, `sa.key`);
 - Корневой CA для extension API-серверов (`front-proxy-ca.key`, `front-proxy-ca.crt`).
2. **Клиентские сертификаты**, необходимые для взаимного подключения компонентов control plane (например, `apiserver.crt`, `apiserver-etcd-client.crt` и т. д.). Эти файлы хранятся только на узлах. При любом изменении (например, добавлении новых SAN) сертификаты автоматически перевыпускаются, а kubeconfig синхронизируется.

Управление PKI

DKP также управляет инфраструктурой частных ключей (PKI), которая необходима для шифрования и аутентификации во всём кластере Kubernetes:

- PKI для компонентов control plane (kube-apiserver, kube-controller-manager, kube-scheduler и т. д.).
- PKI для кластера etcd (сертификаты etcd и межузловое взаимодействие).

DKP «забирает» управление этими PKI после завершения первоначальной установки кластера и запуска своих подов. Таким образом, все операции по выпуску, продлению и обновлению ключей (как для control plane, так и для etcd) выполняются автоматически и централизованно, без необходимости ручного вмешательства.

Дополнительные SAN и автоматическое обновление

Deckhouse упрощает добавление новых точек входа (SAN) для API Kubernetes: достаточно прописать их в конфигурации. После любого изменения в SAN модуль автоматически регенерирует сертификаты и обновляет kubeconfig.

Чтобы добавить дополнительные SAN (дополнительные DNS-имена или IP-адреса) для API Kubernetes пропишите новые SAN в параметре `spec.settings.apiserver.certSANs` вашего ресурса `ModuleConfig/control-plane-manager`.

DKP автоматически сгенерирует новые сертификаты и обновит все необходимые конфигурационные файлы (включая kubeconfig).

Ротация сертификатов kubelet

В Deckhouse Kubernetes Platform ротация сертификатов kubelet происходит автоматически. Параметры `--tls-cert-file` и `--tls-private-key-file` для kubelet не задаются напрямую. Вместо этого используется механизм динамических TLS-сертификатов: kubelet применяет клиентский сертификат по пути `/var/lib/kubelet/pki/kubelet-client-current.pem`, с помощью которого он запрашивает у kube-apiserver новый клиентский или серверный сертификат (файл `/var/lib/kubelet/pki/kubelet-server-current.pem`).

Из-за использования динамических TLS механизмов, модуль `operator-trivy` исключает проверки CIS Benchmark AVD-KCV-0088 и AVD-KCV-0089, так как они предполагают явную передачу параметров `--tls-cert-file` и `--tls-private-key-file` в конфигурации kubelet.

Особенности ротации сертификатов kubelet в Deckhouse Kubernetes Platform:

- По умолчанию kubelet генерирует собственные ключи в каталоге `/var/lib/kubelet/pki/` и при необходимости самостоятельно запрашивает продление сертификатов у kube-apiserver.
- Срок действия выданных сертификатов составляет 1 год (8760 часов). Когда до истечения срока действия остаётся от 5 до 10% времени (точное значение выбирается случайным образом из этого диапазона), kubelet автоматически инициирует запрос на новый сертификат. Подробнее см. в официальной документации Kubernetes. При необходимости срок действия сертификатов можно изменить с помощью параметра `--cluster-signing-duration` в манифесте `/etc/kubernetes/manifests/kube-controller-manager.yaml`. Однако, чтобы kubelet успел получить и установить новый сертификат до истечения текущего, рекомендуется устанавливать срок действия сертификатов не менее чем на 1 час.
- Если kubelet не успеет обновить сертификат до его истечения, он утратит возможность выполнять запросы к kube-apiserver и, соответственно, продлевать сертификаты. В результате узел будет помечен как `NotReady` и автоматически пересоздан.

Ручное обновление сертификатов компонентов control plane

Если master-узлы долго не были доступны (к примеру, серверы были выключены), возможна ситуация, когда некоторые сертификаты в управляющем слое утрачивают актуальность. После включения узлов автоматического

обновления не произойдёт — нужно выполнить процедуру вручную.

Обновление сертификатов компонентов управляющего слоя происходит с помощью утилиты `kubeadm`. Чтобы обновить сертификаты, выполните следующие действия на каждом master-узле:

1. Найдите утилиту `kubeadm` на master-узле и создайте символическую ссылку с помощью следующей команды:

```
ln -s $(find /var/lib/containerd -name kubeadm -type f -executable -print -quit) /usr/bin/kubeadm
```

2. Выполните команду:

```
kubeadm certs renew all
```

Она пересоздаст нужные сертификаты (kube-apiserver, kube-controller-manager, kube-scheduler, etcd и т.д.).

Ускорение перезапуска подов при потере связи с узлом

По умолчанию, если узел в течении 40 секунд не сообщает свое состояние, он помечается как недоступный (Unreachable). И еще через 5 минут поды узла начнут перезапускаться на других узлах. В итоге общее время недоступности приложений составляет около 6 минут.

В специфических случаях, когда приложение не может быть запущено в нескольких экземплярах, есть способ сократить период их недоступности:

1. Уменьшить время перехода узла в состояние `Unreachable` при потере с ним связи настройкой параметра `nodeMonitorGracePeriodSeconds`.
2. Установить меньший таймаут удаления подов с недоступного узла в параметре `failedNodePodEvictionTimeoutSeconds`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```



Чем короче таймауты, тем чаще системные компоненты проверяют состояние узлов и планируют перемещение подов. Это повышает нагрузку на control plane, поэтому выбирайте значения, соответствующие вашим требованиям к отказоустойчивости и производительности.

Принудительное отключение IPv6 на узлах кластера

Внутреннее взаимодействие между компонентами кластера DKP осуществляется по протоколу IPv4. Однако, на уровне операционной системы узлов кластера, как правило, по умолчанию активен IPv6. Это приводит к автоматическому присвоению IPv6-адресов всем сетевым интерфейсам, включая интерфейсы подов. В результате возникает нежелательный сетевой трафик — например, избыточные DNS-запросы типа AAAA, которые могут повлиять на производительность и усложнить отладку сетевых взаимодействий.

Для корректного отключения IPv6 на уровне узлов в кластере, управляемом DKP, достаточно задать необходимые параметры через ресурс `NodeGroupConfiguration`:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: disable-ipv6.sh
spec:
  nodeGroups:
    - '*'
  bundles:
    - '*'
  weight: 50
  content: |
    GRUB_FILE_PATH="/etc/default/grub"

    if ! grep -q "ipv6.disable" "$GRUB_FILE_PATH"; then
      sed -E -e 's/^(GRUB_CMDLINE_LINUX_DEFAULT="[^\"]*)" /\1 ipv6.disable=1/' -i "$GRUB_FILE_PATH"
      update-grub

      bb-flag-set reboot
    fi
```



После применения ресурса настройки GRUB будут обновлены, и узлы кластера начнут последовательную перезагрузку для применения изменений.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `console`
- Документация модуля `control-plane-manager`
- Кастомные ресурсы модуля `node-manager`
- Документация модуля `operator-trivy`
- Документация модуля `user-authz`

Восстановление и отладка control plane

Восстановление при ошибках

В процессе работы DKP автоматически создает резервные копии конфигурации и данных, которые могут пригодиться в случае возникновения проблем. Эти резервные копии сохраняются в директории `/etc/kubernetes/deckhouse/backup`. Если в процессе работы возникли ошибки или непредвиденные ситуации, вы можете использовать эти резервные копии для восстановления до предыдущего исправного состояния.

Восстановление работоспособности кластера etcd

Если кластер etcd не функционирует и не удается восстановить его из резервной копии, вы можете попытаться восстановить его с нуля, следуя шагам ниже.

1. Сначала на всех узлах, которые являются частью вашего кластера etcd, **кроме одного**, удалите манифест `etcd.yaml`, который находится в директории `/etc/kubernetes/manifests/`. После этого только один узел останется активным, и с него будет происходить восстановление состояния мультимастерного кластера.
2. На оставшемся узле откройте файл манифеста `etcd.yaml` и укажите параметр `--force-new-cluster` в `spec.containers.command`.
3. После успешного восстановления кластера, удалите параметр `--force-new-cluster`.



Эта операция является деструктивной, так как она полностью уничтожает текущие данные и инициализирует кластер с состоянием, которое сохранено на узле. Все pending-записи будут утеряны.

Восстановление master-узла при ошибке загрузки компонентов control plane через kubelet

Ситуация, когда kubelet не может загрузить компоненты control plane, может возникнуть, если в кластере с одним master-узлом был удален каталог с образами (например, `/var/lib/containerd`). В этом случае после перезапуска kubelet не сможет загрузить образы компонентов control plane, поскольку параметры авторизации для доступа к `registry.deckhouse.ru` на master-узле отсутствуют.

Далее приведена инструкция по восстановлению master-узла.

containerd

1. Для восстановления работоспособности master-узла нужно в любом рабочем кластере под управлением DKP выполнить команду:

```
d8 k -n d8-system get secrets deckhouse-registry -o json |
jq -r '.data[".dockerconfigjson"]' | base64 -d |
jq -r '.auths["registry.deckhouse.ru"].auth'
```

2. Скопируйте вывод команды и присвойте переменной `AUTH` на поврежденном master-узле.
3. Далее на поврежденном master-узле загрузите образы компонентов control plane:

```
for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
  crictl pull --auth $AUTH $image
done
```

4. После загрузки образов перезапустите kubelet.

Восстановление etcd

Просмотр списка узлов кластера в etcd

Способ 1

Используйте команду `etcdctl member list`.

Пример:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Внимание. Последний параметр в таблице вывода показывает, что узел находится в состоянии `learner`, а не в состоянии `leader`.

Способ 2

Используйте команду `etcdctl endpoint status`. Для лидера в столбце `IS LEADER` будет указано значение `true`.

Пример:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Восстановление кластера etcd при полной недоступности

1. Остановите все узлы etcd, кроме одного, удалив манифест `etcd.yaml` на остальных.
2. На оставшемся узле добавьте в команду etcd параметр `--force-new-cluster`.
3. После восстановления удалите этот флаг.

✗ Будьте осторожны: эти действия полностью уничтожают предыдущие данные и формируют новый кластер etcd.

Восстановление etcd при ошибке `panic: unexpected removal of unknown remote peer`

1. Найдите утилиту `etcdctl` на master-узле и скопируйте исполняемый файл в `/usr/local/bin/`:

```
cp $(find /var/lib/containerd/ \
-name etcdctl -print -quit) /usr/local/bin/etcdctl
```

2. Создайте новый снимок базы etcd на основе текущего локального снимка (`/var/lib/etcd/member/snap/db`):

```
etcdctl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
--initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-urls=https://<ADDRESS>:2380 \
--skip-hash-check=true --data-dir /var/lib/etcdtest
```

где:

- `<HOSTNAME>` — имя master-узла;
- `<ADDRESS>` — адрес master-узла.

3. Выполните следующие команды для использования нового снимка:

```
cp -r /var/lib/etcd /tmp/etcd-backup
rm -rf /var/lib/etcd
mv /var/lib/etcdtest /var/lib/etcd
```

4. Найдите контейнеры `etcd` и `api-server`:

```
crictl ps -a | egrep "etcd|apiserver"
```

5. Удалите найденные контейнеры `etcd` и `api-server`:

```
crictl rm <CONTAINER-ID>
```

6. Перезапустите master-узел.

Действия при переполнении базы данных etcd (превышение `quota-backend-bytes`)

Когда объем базы данных etcd достигает лимита, установленного параметром `quota-backend-bytes`, доступ к ней становится read-only. Это означает, что база данных etcd перестает принимать новые записи, но при этом остается доступной для чтения данных. Вы можете понять, что столкнулись с подобной ситуацией, выполнив команду:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ endpoint status -w table --cluster
```

Если в поле `ERRORS` вы видите подобное сообщение `alarm:NOSPACE`, значит вам нужно предпринять следующие шаги:

1. Найдите строку с `--quota-backend-bytes` в файле манифеста пода etcd, расположенного по пути `/etc/kubernetes/manifests/etcd.yaml` и увеличьте значение, умножив указанный параметр в этой строке на два. Если такой строки нет — добавьте, например: `--quota-backend-bytes=8589934592`. Эта настройка задает лимит на 8 ГБ.
2. Сбросьте активное предупреждение (alarm) о нехватке места в базе данных. Для этого выполните следующую команду:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ alarm disarm
```

3. Измените параметр `maxDbSize` в настройках модуля `control-plane-manager` на тот, который был задан в манифесте.

Дефрагментация etcd



Перед дефрагментацией создайте резервную копию etcd.

Для просмотра размера БД etcd на определенном узле перед дефрагментацией и после ее выполнения используйте команду (здесь `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
endpoint status --cluster -w table
```

Пример вывода (размер БД etcd на узле указывается в колонке `DB SIZE`):

```
+-----+-----+-----+-----+-----+-----+-----+
|          ENDPOINT          |      ID      | VERSION | STORAGE VERSION | DB SIZE | IN USE |
| PERCENTAGE NOT IN USE | QUOTA  | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED
| INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | true | false | 56 | 258054684 | 258054684 |
| false |
+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |
| false |
+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |
| false |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
```

Как выполнить дефрагментацию etcd узла в кластере с одним master-узлом



Дефрагментация etcd — ресурсоемкая операция, которая на время полностью блокирует работу etcd на данном узле. Учитывайте это при выборе времени для проведения операции в кластере с одним master-узлом.

Чтобы выполнить дефрагментацию etcd в кластере с одним master-узлом, используйте следующую команду (здесь `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Пример вывода при успешном выполнении операции:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```



При появлении ошибки из-за таймаута увеличивайте значение параметра `--command-timeout` из команды выше, пока дефрагментация не выполнится успешно.

Как выполнить дефрагментацию etcd в кластере с несколькими master-узлами

Чтобы выполнить дефрагментацию etcd в кластере с несколькими master-узлами:

1. Получите список подов etcd. Для этого используйте следующую команду:

```
d8 k -n kube-system get pod -l component=etcd -o wide
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED
etcd-master-0	1/1	Running	0	3d21h	192.168.199.80	master-0	<none>
etcd-master-1	1/1	Running	0	3d21h	192.168.199.81	master-1	<none>
etcd-master-2	1/1	Running	0	3d21h	192.168.199.82	master-2	<none>

2. Определите master-узел — лидер. Для этого обратитесь к любому поду etcd и получите список узлов — участников кластера etcd с помощью команды (где `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
endpoint status --cluster -w table
```

Пример вывода (у лидера в колонке `IS LEADER` будет значение `true`):

```

+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
|          ENDPOINT          |          ID          | VERSION | STORAGE VERSION | DB SIZE | IN USE |
| PERCENTAGE NOT IN USE | QUOTA  | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT
APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
|          20% | 2.1 GB |          true |          false | 56 | 258054684 |
258054684 |          |          |          false |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
|          20% | 2.1 GB |          false |          false | 56 | 258054685 |
258054685 |          |          |          false |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
|          20% | 2.1 GB |          false |          false | 56 | 258054685 |
258054685 |          |          |          false |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+

```

3. Поочередно выполните дефрагментацию etcd узлов — участников etcd кластера. Для дефрагментации используйте команду (здесь `NODE_NAME` — имя master-узла):



Важно: дефрагментацию лидера необходимо выполнять в последнюю очередь.

Восстановление etcd на узле после дефрагментации может занять некоторое время. Рекомендуется подождать не менее минуты прежде чем переходить к дефрагментации etcd следующего узла.

```

d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s

```

Пример вывода при успешном выполнении операции:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```



При появлении ошибки из-за таймаута увеличивайте значение параметра `--command-timeout` из команды выше, пока дефрагментация не выполнится успешно.

Отказоустойчивость

Если какой-либо компонент control plane становится недоступным, кластер временно сохраняет текущее состояние, но не может обрабатывать новые события. Например:

- При сбое `kube-controller-manager` перестаёт работать масштабирование deployment'ов.
- При недоступности `kube-apiserver` невозможны любые запросы к Kubernetes API, но уже запущенные приложения продолжают функционировать.

Однако при продолжительной недоступности компонентов нарушается обработка новых объектов, реакция на сбои узлов и другие процессы. Со временем это может привести к деградации работы кластера и повлиять на пользовательские приложения.

Чтобы снизить такие риски, следует масштабировать control plane до отказоустойчивой конфигурации — минимум трёх узлов. Это особенно критично для etcd, так как он требует наличия кворума для выбора лидера. Кворум работает по принципу большинства ($N/2 + 1$) от общего числа узлов.

Пример:

Размер кластера	Большинство	Максимальные потери
1	1	0
3	2	1

Размер кластера	Большинство	Максимальные потери
5	3	2
7	4	3
9	5	4

i Чётное число узлов не даёт преимущества по отказоустойчивости, но увеличивает накладные расходы на репликацию.

В большинстве случаев достаточно трёх узлов etcd, пять — если критична устойчивость. Более семи — крайне редко и не рекомендуется из-за высокой нагрузки.

После добавления новых узлов control plane:

- Устанавливается лейбл `node-role.kubernetes.io/control-plane=""`.
- DaemonSet запускает поды на новых узлах.
- DKP создает или обновляет файлы в `/etc/kubernetes`: манифесты, конфигурации, сертификаты и т.д.
- Все модули DKP с поддержкой отказоустойчивости автоматически включают её, если значение глобальной настройки `highAvailability` не переопределено вручную.

Удаление узлов control plane выполняется в обратном порядке:

- Удаляются лейблы `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master`, `node.deckhouse.io/group`.
- DKP удаляет свои поды с этих узлов.
- Члены etcd, расположенные на этих узлах, удаляются автоматически.
- Если число узлов уменьшается с двух до одного, etcd может перейти в статус `readonly`. В этом случае требуется запуск с параметром `--force-new-cluster`, который следует убрать после успешного запуска.

Дополнительные ресурсы

- Глобальные параметры
- Параметры модуля `control-plane-manager`

Масштабирование и изменение master-узлов

Масштабирование и переход single-master/multi-master

Режимы работы control plane

Deckhouse Kubernetes Platform (DKP) поддерживает два режима работы control plane:

1. **Single-master:**
 - `kube-apiserver` использует только локальный экземпляр etcd;
 - на узле запускается прокси-сервер, который принимает запросы на `localhost`;
 - `kube-apiserver` “слушает” только на IP-адресе master-узла.
2. **Multi-master:**
 - `kube-apiserver` работает со всеми экземплярами etcd в кластере;
 - на всех узлах настраивается дополнительный прокси:
 - если локальный `kube-apiserver` недоступен, запросы автоматически перенаправляются на другие узлы;
 - это обеспечивает отказоустойчивость и возможность масштабирования.

Автоматическое масштабирование master-узлов

Deckhouse Kubernetes Platform (DKP) позволяет автоматически добавлять и удалять master-узлы, используя лейбл `node-role.kubernetes.io/control-plane=""`.

Автоматическое управление master-узлами:

- Добавление лейбла `node-role.kubernetes.io/control-plane=""` на узел:
 - разворачиваются все компоненты control plane;
 - узел подключается к etcd-кластеру;
 - автоматически регенерируются сертификаты и конфигурационные файлы.
- Удаление лейбла `node-role.kubernetes.io/control-plane=""` с узла:
 - компоненты control plane удаляются;
 - узел корректно исключается из etcd-кластера;
 - обновляются связанные конфигурационные файлы.

i Переход с 2 master-узлов до 1 требует ручной корректировки etcd. В остальных случаях изменение количества master-узлов выполняется автоматически.

Типовые сценарии масштабирования

⚠ Для стабильности кластера необходимо поддерживать нечётное число узлов с etcd для обеспечения кворума.

Deckhouse Kubernetes Platform (DKP) поддерживает автоматическое и ручное масштабирование master-узлов как в облачных, так и в bare-metal кластерах:

1. Миграция single-master → multi-master:

- добавьте один или несколько новых master-узлов;
- установите им лейбл `node-role.kubernetes.io/control-plane=""`;
- DKP автоматически:
 - развернёт все компоненты control plane;
 - настроит узлы для работы с etcd-кластером;
 - синхронизирует сертификаты и конфигурационные файлы.

2. Миграция multi-master → single-master:

- снимите лейблы `node-role.kubernetes.io/control-plane=""` и `node-role.kubernetes.io/master=""` со всех лишних master-узлов;
- для **bare-metal кластеров**:
 - чтобы корректно исключить узлы из etcd:
 - выполните команду `d8 k delete node <имя-узла>`;
 - выключите соответствующие виртуальные машины или серверы.

Важно. В облачных кластерах все необходимые действия автоматически выполняются с помощью команды `dhctl converge`.

3. Изменение числа master-узлов в облачном кластере:

- Аналогично добавлению/удалению узлов, но чаще всего выполняется с помощью команды `dhctl converge` или вручную через облачные инструменты.

4. Миграция multi-master (3 master-узла) → 2 master-узла и 1 arbiter-узел в облачном кластере

- В настройках облачного провайдера для параметра `masterNodeGroup.replicas` укажите 2 и создайте NodeGroup для arbiter-узла. Аналогично добавлению/удалению узлов, но чаще всего выполняется с помощью команды `dhctl converge` или вручную через облачные инструменты.

Подробнее о настройке режима HA с двумя master-узлами и arbiter-узлом в облачном кластере можно почитать в разделе Управление режимом HA.

5. Миграция multi-master (3 master-узла) → 2 master-узла и 1 arbiter-узел в статическом кластере

- создайте NodeGroup для arbiter-узла и добавьте узел в кластер;
- снимите лейблы `node-role.kubernetes.io/control-plane=""`, `node-role.kubernetes.io/master=""` и `node.deckhouse.io/group=""` с лишнего master-узла;
- чтобы корректно исключить узел из etcd в **bare-metal кластере**:
 - выполните команду `d8 k delete node <имя-узла>`;
 - выключите соответствующую виртуальную машину или сервер.

Подробнее о настройке режима HA с двумя master-узлами и arbiter-узлом в статическом кластере можно почитать в разделе Управление режимом HA.

Удаление роли master с узла без удаления самого узла



Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

Если необходимо вывести узел из состава master-узлов, но сохранить его в кластере для других задач, выполните следующие шаги:

1. Снимите лейблы, чтобы узел больше не рассматривался как master:

```
d8 k label node <имя-узла> node-role.kubernetes.io/control-plane-
d8 k label node <имя-узла> node-role.kubernetes.io/master-
d8 k label node <имя-узла> node.deckhouse.io/group-
```

2. Убедитесь, что удаляемый master-узел пропал из списка членов кластера etcd:

Пример:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

3. Удалите статические манифесты компонентов control plane, чтобы они больше не запускались на узле и лишние файлы PKI. Для этого зайдите на узел и выполните следующие команды:


```
rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
rm -f /etc/kubernetes/authorization-webhook-config.yaml
rm -f /etc/kubernetes/admin.conf /root/.kube/config
rm -rf /etc/kubernetes/deckhouse
rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
rm -rf /var/lib/etcd/member/
```

После выполнения этих шагов узел больше не будет считаться master-узлом, но останется в кластере и может использоваться для других задач.

Изменение образа ОС master-узлов в мультимастерном кластере

Способ изменения ОС зависит от типа кластера: в облачном кластере узлы заменяются через `dhctl converge`, в статическом — вручную, по одному узлу.

В облачном кластере

Чтобы изменить образ ОС master-узлов в облачном мультимастерном кластере, выполните следующие шаги:



Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
4. Убедитесь, что очередь Deckhouse пуста.

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.



При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду, чтобы проверить состояние перед началом работы:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Ответ должен сообщить, что Terraform не нашёл расхождений и изменений не требуется.

8. **В контейнере с инсталлятором** выполните следующую команду и укажите необходимый образ ОС в параметре `masterNodeGroup.instanceClass` (укажите адреса всех master-узлов в параметре `--ssh-host`):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

9. **В контейнере с инсталлятором** выполните следующую команду, чтобы провести обновление узлов:

Внимательно изучите действия, которые планирует выполнить `converge`, когда запрашивает подтверждение.

При выполнении команды узлы будут заменены на новые с подтверждением на каждом узле. Замена будет выполняться по очереди в обратном порядке (2,1,0).

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Следующие действия **выполняйте поочерёдно на каждом** master-узле, начиная с узла с наивысшим номером (с суффиксом 2) и заканчивая узлом с наименьшим номером (с суффиксом 0).

10. На созданном узле откройте журнал systemd-юнита `bashible.service`. Дождитесь окончания настройки узла — в журнале должно появиться сообщение `nothing to do`:

```
journalctl -fu bashible.service
```

11. Проверьте, что узел `etcd` отобразился в списке узлов кластера:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

12. Убедитесь, что `control-plane-manager` функционирует на узле.

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
-l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
```

13. Выполните проверку очереди Deckhouse и убедитесь, что отсутствуют ошибки, с помощью команды:

```
d8 system queue list
```

14. Перейдите к обновлению следующего узла.

В статическом кластере

Инструкция ниже описывает замену ОС на master-узлах, которые добавлены в кластер **вручную** с помощью скрипта `bootstrap.sh`. Выполняйте шаги поочерёдно для каждого master-узла и переходите к следующему только после того, как текущий узел вернулся в кластер и стал работоспособен.

⚠ Если master-узлы управляются Cluster API Provider Static (CAPS) через ресурсы `StaticInstance`, не используйте эту инструкцию. Сначала удалите `StaticInstance`, установите требуемую ОС и снова добавьте узел в `NodeGroup master`.

⚠ Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

Чтобы изменить ОС вручную добавленного master-узла, выполните следующие шаги:

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`. Если используется модуль `stronghold`, убедитесь, что создана резервная копия его данных.
2. Проверьте состояние кластера, отсутствие алертов и незавершённых задач в очереди Deckhouse:

```
d8 status
```

3. Снимите с узла лейблы master-узла:

```
d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
node-role.kubernetes.io/master- node.deckhouse.io/group-
```

где `<MASTER_NODE_NAME>` — имя изменяемого master-узла.

4. Убедитесь, что узел удалён из списка членов кластера `etcd`:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

5. Вытесните нагрузку с узла командой `d8 k drain`:

```
d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
```

6. Принудительно удалите оставшиеся на узле поды:

```
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --force
```

7. Удалите объект Node:

```
d8 k delete node <MASTER_NODE_NAME>
```

8. На удаляемом master-узле очистите данные DKP. Команда необратимо удаляет данные Kubernetes и DKP с узла. Перед выполнением убедитесь, что выбран правильный узел и созданы необходимые резервные копии:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

9. Установите на узле требуемую ОС.

10. На одном из оставшихся в кластере master-узлов получите и раскодируйте скрипт для добавления master-узла:

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
  -o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
```

11. Безопасно скопируйте полученный на предыдущем шаге файл `bootstrap.sh` на добавляемый узел и выполните его на этом узле от пользователя `root`:

```
bash bootstrap.sh
```

12. **На добавляемом узле** откройте журнал systemd-юнита `bashible.service`. Дождитесь окончания настройки узла — в журнале должно появиться сообщение `nothing to do`:

```
journalctl -fu bashible.service
```

13. Дождитесь перехода узла в статус `Ready`:

```
d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
```

14. Проверьте, что узел отобразился в списке членов кластера etcd:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

15. Убедитесь, что `control-plane-manager` функционирует на узле:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
  -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
```

16. Убедитесь, что в кластере нет алертов и незавершённых задач:

```
d8 status
```

17. Повторите процедуру для следующего master-узла.

Изменение образа ОС в кластере с одним master-узлом

Способ зависит от типа кластера: сначала добавьте дополнительные master-узлы, замените ОС в мультимастерном режиме, затем верните исходное число master-узлов.

⚠ Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

В облачном кластере

1. Преобразуйте кластер с одним master-узлом в мультимастерный в соответствии с инструкцией.
2. Измените ОС на master-узлах в соответствии с инструкцией.
3. Преобразуйте мультимастерный кластер в кластер с одним master-узлом в соответствии с инструкцией.

В статическом кластере

1. Добавьте дополнительные master-узлы в соответствии с инструкцией.
2. Измените ОС на master-узлах в соответствии с инструкцией.
3. Выведите лишние master-узлы из роли control plane в соответствии с инструкцией. Затем удалите их из кластера командой `d8 k delete node <MASTER_NODE_NAME>` и выключите соответствующие серверы.

Добавление master-узлов в статический или гибридный кластер

i Важно иметь нечетное количество master-узлов для обеспечения кворума.

⚠ Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

В процессе установки Deckhouse Kubernetes Platform с настройками по умолчанию в NodeGroup `master` отсутствует секция `spec.staticInstances.labelSelector` с настройками фильтра лейблов по ресурсам `staticInstances`. Из-за этого после изменения количества узлов `staticInstances` в NodeGroup `master` (параметр `spec.staticInstances.count`) при добавлении обычного узла с помощью Cluster API Provider Static (CAPS) он может быть «перехвачен» и добавлен в NodeGroup `master`, даже если в соответствующем ему `StaticInstance` (в `metadata`) указан лейбл с `role`, отличающийся от `master`. Чтобы избежать этого «перехвата», после установки DKP измените NodeGroup `master` — добавьте в нее секцию `spec.staticInstances.labelSelector` с настройками фильтра лейблов по ресурсам `staticInstances`. Пример NodeGroup `master` с `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Далее при добавлении в кластер master-узлов с помощью CAPS указывайте в соответствующих им `StaticInstance` лейбл, заданный в `spec.staticInstances.labelSelector` NodeGroup `master`. Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # Лейбл, указанный в spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Укажите IP-адрес сервера статического узла.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

- i** При добавлении новых master-узлов с помощью CAPS и изменении в NodeGroup `master` количества master-узлов (параметр `spec.staticInstances.count`) учитывайте следующее:

При первоначальной установке кластера в конфигурации указывается первый master-узел, на который происходит установка. Если после установки нужно сделать мультимастер и добавить master-узлы с помощью CAPS, в параметре `spec.staticInstances.count` NodeGroup `master` необходимо указать количество узлов на один меньше желаемого.

Например, если нужно сделать мультимастер с тремя master-узлами в `spec.staticInstances.count` NodeGroup `master` укажите значение 2 и создайте два `staticInstances` для добавляемых узлов. После их добавления в кластер количество master-узлов будет равно трём: master-узел, на который происходила установка и два master-узла, добавленные с помощью CAPS.

В остальном добавление master-узла в статический или гибридный кластер аналогично добавлению обычного узла. Воспользуйтесь для этого соответствующими примерами. Все необходимые действия по настройке компонентов control plane кластера на новом узле будут выполнены автоматически, дождитесь их завершения — появления master-узлов в статусе `Ready`.

Добавление master-узлов в облачном кластере

Далее описана конвертация кластера с одним master-узлом в мультимастерный кластер.

- ⚠** Перед добавлением узлов убедитесь в наличии необходимых квот. Важно иметь нечетное количество master-узлов для обеспечения кворума.

- ⚠** Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать созданию новых master-узлов.
4. Убедитесь, что очередь Deckhouse пуста:

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.

- i** При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду, чтобы проверить состояние перед началом работы:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST>
```

Ответ должен сообщить, что Terraform не нашёл расхождений и изменений не требуется.

8. **В контейнере с инсталлятором** выполните следующую команду и укажите требуемое количество master-узлов в параметре `masterNodeGroup.replicas`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST>
```

i Для **Yandex Cloud**, при использовании внешних адресов на master-узлах, количество элементов массива в параметре `masterNodeGroup.instanceClass.externalIPAddresses` должно равняться количеству master-узлов. При использовании значения `Auto` (автоматический заказ публичных IP-адресов), количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при трех master-узлах (`masterNodeGroup.replicas: 3`) и автоматическом заказе адресов, параметр `masterNodeGroup.instanceClass.externalIPAddresses` будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
- "Auto"
- "Auto"
```

9. В контейнере с инсталлятором выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

10. Дождитесь появления необходимого количества master-узлов в статусе `Ready` и готовности всех экземпляров `control-plane-manager`:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-control-plane-manager
```

Уменьшение числа master-узлов в облачном кластере

Далее описана конвертация мультимастерного кластера в кластер с одним master-узлом.

 Описанные ниже шаги необходимо выполнять с первого по порядку master-узла кластера (`master-0`). Это связано с тем, что кластер всегда масштабируется по порядку: например, невозможно удалить узлы `master-0` и `master-1`, оставив `master-2`.

 Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
4. Убедитесь, что очередь DKP пуста:

```
d8 system queue list
```

5. На локальной машине авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.

i При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. На локальной машине запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. В контейнере с инсталлятором выполните следующую команду и укажите `1` в параметре `masterNodeGroup.replicas`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
--ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

i Для **Yandex Cloud** при использовании внешних адресов на master-узлах количество элементов массива в параметре `masterNodeGroup.instanceClass.externalIPAddresses` должно равняться количеству master-узлов. При использовании значения `Auto` (автоматический заказ публичных IP-адресов) количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при одном master-узле (`masterNodeGroup.replicas: 1`) и автоматическом заказе адресов параметр `masterNodeGroup.instanceClass.externalIPAddresses` будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
```

8. В контейнере с инсталлятором выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST>
```

i Для **OpenStack** и **VK Cloud (OpenStack)** после подтверждения удаления узла обязательно проверьте удаление диска `<prefix>kubernetes-data-N` в самом Openstack.

Например, при удалении узла `cloud-demo-master-2` в веб-интерфейсе Openstack или в OpenStack CLI необходимо проверить отсутствие диска `cloud-demo-kubernetes-data-2`.

В случае, если диск `kubernetes-data` останется, при увеличении количества master-узлов могут возникнуть проблемы в работе ETCD.

9. Выполните проверку очереди Deckhouse и убедитесь, что отсутствуют ошибки, с помощью команды:

```
d8 system queue list
```

Доступ к контроллеру DKP в мультимастерном кластере

В кластерах с несколькими master-узлами DKP запускается в режиме высокой доступности (в нескольких экземплярах). Для доступа к активному контроллеру DKP можно использовать следующую команду (на примере команды `deckhouse-controller queue list`):

```
d8 system queue list
```

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Кастомные ресурсы модуля `node-manager`
- Документация модуля `stronghold`

Основы управления узлами в Deckhouse

Deckhouse Kubernetes Platform (DKP) поддерживает полный цикл управления узлами:

- Автоматическое масштабирование количества узлов в зависимости от нагрузки;
- Обновление узлов и поддержание их в актуальном состоянии;
- Централизованное управление конфигурацией групп узлов через CRD `NodeGroup`;
- Использование различных типов узлов: постоянные, временные, облачные или `bare-metal`.

i DKP может работать как с `bare-metal`, так и с облачными кластерами, обеспечивая гибкость и расширяемость.

Группы узлов позволяют логически сегментировать инфраструктуру кластера. В DKP часто используются следующие типы `NodeGroup` по назначению:

- `master` — управляющие узлы (control plane);
- `front` — узлы для маршрутизации HTTP(S)-трафика;
- `monitoring` — узлы для размещения компонентов мониторинга;
- `worker` — узлы для пользовательских приложений;
- `system` — выделенные узлы для системных компонентов.

В каждой группе можно централизованно задавать настройки узлов, включая версию Kubernetes, ресурсы, `taint`'ы, лейблы, параметры `kubelet` и прочее.

i Управление узлами доступно только для кластеров, полностью управляемых DKP. Для Managed Kubernetes-кластеров такая возможность не поддерживается.

Автоматическое развёртывание и обновление

В Deckhouse Kubernetes Platform (DKP) реализован автоматизированный механизм управления жизненным циклом узлов на основе объектов NodeGroup. DKP обеспечивает как начальное развёртывание узлов, так и их обновление при изменении конфигурации, поддерживая как облачные, так и bare-metal кластеры (при наличии интеграции с модулем `node-manager`).

Как это работает:

1. NodeGroup — основной объект управления группами узлов. Он определяет тип узлов, их количество, шаблоны ресурсов и ключевые параметры (например, настройки kubelet, taint'ов и др.).
2. При создании или изменении NodeGroup, модуль `node-manager` автоматически приводит состояние узлов в соответствие с заданной конфигурацией.
3. Обновление происходит без вмешательства пользователя — устаревшие узлы удаляются, новые создаются автоматически.

Рассмотрим автоматическое обновление на примере обновления версии kubelet.

1. Пользователь изменяет параметры в секции `kubelet` объекта NodeGroup.
2. DKP определяет, что текущие узлы не соответствуют новой конфигурации.
3. Последовательно создаются новые узлы с обновлёнными настройками.
4. Старые узлы постепенно удаляются из кластера.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-cloud
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: AnotherCloudInstanceClass
      name: my-class
```

Базовая настройка узлов и операционной системы

При создании и подключении узлов DKP автоматически выполняет ряд действий, необходимых для корректной работы кластера:

- установка и настройка поддерживаемой операционной системы;
- отключение автоматических обновлений пакетов;
- настройка журналирования и системных параметров;
- установка необходимых пакетов и утилит;
- настройка компонента `kubernetes-api-proxy` для балансировки трафика от kubelet к API-серверам;
- установка и конфигурация компонентов container runtime (`containerd`) и `kubelet`;
- включение узла в состав кластера Kubernetes.

Эти операции выполняются автоматически при использовании `bootstrap.sh` или при подключении узлов через ресурсы StaticInstance и SSHCredentials.

Обновления, требующие прерывания работы узла

Некоторые обновления, например, обновление версии `containerd` или обновление kubelet на несколько версий вперед, требуют прерывания работы узла и могут привести к кратковременному простоем системных компонентов (*disruptive-обновления*). Режим применения таких обновлений настраивается с помощью параметра `disruptions.approvalMode`:

- **Manual** — режим ручного подтверждения disruptive-обновлений. При появлении доступного disruptive-обновления отображается алерт `NodeRequiresDisruptionApprovalForUpdate`.

Чтобы подтвердить disruptive-обновление, установите аннотацию `update.node.deckhouse.io/disruption-approved=` на каждый узел в группе, следуя примеру:

```
d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=
```

i

Важно. В этом режиме не выполняется автоматический drain узла. При необходимости выполните drain вручную перед установкой аннотации.

Чтобы избежать проблем при выполнении drain, всегда устанавливайте режим `Manual` для группы master-узлов.

- **Automatic** — режим автоматического разрешения disruptive-обновлений.

В этом режиме по умолчанию выполняется автоматический drain узла перед применением обновления. Поведение можно изменить с помощью параметра `disruptions.automatic.drainBeforeApproval` в настройках узла.

- **RollingUpdate** — режим, при котором будет создан новый узел с обновлёнными настройками, а старый будет удалён. Применим только к облачным узлам.

В этом режиме на время обновления в кластере создаётся дополнительный узел. Это может быть удобно, если в кластере нет ресурсов для временного размещения нагрузки с обновляемого узла.

Пример системной NodeGroup

Системные узлы — это узлы, предназначенные для запуска системных компонентов. Обычно они выделяются с помощью лейблов и taint'ов, чтобы туда не попадали пользовательские поды. Системные узлы могут быть как статическими, так и облачными.

Пример:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  nodeType: Static
```

Примеры описания NodeGroupConfiguration

Установка плагина cert-manager для kubectl на master-узлах

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
    - "*"
  nodeGroups:
    - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-manager/releases/download/v1.7.1/kubectl-
    cert_manager-linux-amd64.tar.gz -o - | tar -zxvf - kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Конфигурация werf для игнорирования состояния Ready у группы узлов

Werf проверяет состояние Ready у ресурсов и в случае его наличия дожидается, пока значение станет True.

Создание (обновление) ресурса NodeGroup в кластере может занять значительное время (до готовности всех узлов). При использовании werf (например, в CI/CD) это может привести к превышению таймаута сборки.

Чтобы werf игнорировал состояние NodeGroup, добавьте к ресурсу NodeGroup следующие аннотации:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

Настройки для групп с узлами Static и CloudStatic

Группы узлов с типами Static и CloudStatic предназначены для управления вручную созданными узлами — как физическими (bare-metal), так и виртуальными (в облаке, но без участия автоматических контроллеров DKP). Эти узлы подключаются вручную или через StaticInstance и не поддерживают автоматическое обновление и масштабирование.

Особенности конфигурации:

- Все действия по обновлению (обновление kubelet, перезапуск, замена узлов) выполняются вручную или через внешние автоматизации вне DKP.
- Рекомендуется явно указывать желаемую версию kubelet, чтобы обеспечить единообразие между узлами, особенно если они подключаются с разными версиями вручную:

```
nodeTemplate:
  kubelet:
    version: "1.28"
```

- Подключение узлов к кластеру может выполняться вручную или автоматически, в зависимости от конфигурации:
 - **Вручную** — пользователь скачивает bootstrap-скрипт, настраивает сервер, запускает скрипт вручную.
 - **Автоматически (CAPS)** — при использовании StaticInstance и SSHCredentials, DKP автоматически подключает и настраивает узлы.
 - **Смешанный подход** — вручную добавленный узел можно передать под управление CAPS, используя аннотацию `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Если включён Cluster API Provider Static (CAPS), в NodeGroup можно использовать секцию `staticInstances`. Это позволяет DKP автоматически подключать, настраивать и, при необходимости, отключать статические узлы на основе ресурсов StaticInstance и SSHCredentials.

i В NodeGroup с типами Static и CloudStatic можно явно указать количество узлов в параметре `spec.staticInstances.count`. Это позволяет задать ожидаемое количество узлов — DKP использует это значение для контроля состояния и автоматизации.

Запуск DKP на произвольном узле

Для запуска DKP на произвольном узле установите у модуля `deckhouse` соответствующий параметр `nodeSelector` и не задавайте `tolerations`. Необходимые значения `tolerations` в этом случае будут проставлены автоматически.

 Используйте для запуска DKP только узлы с типом **CloudStatic** или **Static**. Также избегайте использования для запуска DKP группы узлов (NodeGroup), содержащей только один узел.

Пример конфигурации модуля:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    nodeSelector:
      node-role.deckhouse.io/deckhouse: ""
```

Дополнительные ресурсы

- Параметры модуля `deckhouse`
- Кастомные ресурсы модуля `node-manager`

Добавление и управление облачными узлами

В Deckhouse Kubernetes Platform (DKP) облачные узлы могут быть следующих типов:

- **CloudEphemeral** — временные, автоматически создаваемые и удаляемые узлы;
- **CloudPermanent** — постоянные узлы, управляемые вручную через `replicas`;
- **CloudStatic** — статические облачные узлы. Машины создаются вручную или внешними средствами, а DKP подключает их к кластеру и управляет ими как обычными узлами.

Ниже приведены инструкции по добавлению и настройке каждого типа.

Добавление CloudEphemeral-узлов в облачном кластере

CloudEphemeral-узлы автоматически создаются и управляются в кластере с помощью Machine Controller Manager (MCM) или Cluster API (в зависимости от конфигурации) — оба компонента входят в состав модуля `node-manager` в DKP.

Для добавления узлов:

1. Убедитесь, что включён модуль облачного провайдера. Например: `cloud-provider-yandex`, `cloud-provider-openstack`, `cloud-provider-aws`.
2. Создайте объект `InstanceClass` с конфигурацией машин. Этот объект описывает параметры виртуальных машин, создаваемых в облаке:

Пример (для OpenStack):

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: worker-instance
spec:
  flavorName: m1.medium
  imageName: ubuntu-24-04-cloud-amd64
  rootDiskSize: 20
  mainNetwork: default
```

Здесь задаются:

- `flavorName` — тип инстанса (CPU/RAM);
- `imageName` — образ ОС;
- `rootDiskSize` — размер системного диска;
- `mainNetwork` — облачная сеть для инстанса.

3. Создайте `NodeGroup` с типом `CloudEphemeral`. Пример манифеста:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: worker-instance
    minPerZone: 1
    maxPerZone: 3
    zones:
      - nova
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
    taints: []
```

4. Дождитесь запуска и автоматического добавления узлов.

Настройки для групп с узлами CloudEphemeral

Группы узлов с типом CloudEphemeral предназначены для автоматического масштабирования за счёт создания и удаления виртуальных машин в облаке с помощью Machine Controller Manager (MCM). Этот тип групп широко применяется в облачных кластерах DKP.

Конфигурация узлов задаётся в секции `cloudInstances` и включает параметры для масштабирования, зонирования, резервирования и приоритизации.

Пример базовой конфигурации:

```
спес:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1
    maxPerZone: 5
    maxUnavailablePerZone: 1
    zones:
      - ru-central1-a
      - ru-central1-b
```

Изменение конфигурации облачного провайдера в кластере

Настройки используемого облачного провайдера в облачном или гибридном кластере хранятся в структуре `<PROVIDER_NAME>ClusterConfiguration`, где `<PROVIDER_NAME>` — название/код провайдера. Например, для провайдера OpenStack структура будет называться `OpenStackClusterConfiguration`.

Независимо от используемого облачного провайдера его настройки можно изменить с помощью следующей команды:

```
d8 system edit provider-cluster-configuration
```

Автомасштабирование группы узлов

В Deckhouse Kubernetes Platform (DKP) автомасштабирование группы узлов происходит на основе потребностей в ресурсах (CPU и память) и выполняется компонентом `Cluster Autoscaler`, входящим в модуль `node-manager`.

Автоматическое масштабирование происходит только при наличии Pending-подов, которые не могут быть запущены на существующих узлах из-за нехватки ресурсов (например, CPU или памяти). В этом случае `Cluster Autoscaler` пытается добавить узлы, основываясь на конфигурации `NodeGroup`.

Основные параметры масштабирования задаются в секции `cloudInstances` ресурса `NodeGroup`:

- `minPerZone` — минимальное количество виртуальных машин в каждой зоне. Это число всегда поддерживается даже при отсутствии нагрузки.
- `maxPerZone` — максимальное количество узлов, которые можно создать в каждой зоне. Определяет верхнюю границу масштабирования.
- `maxUnavailablePerZone` — ограничивает количество недоступных узлов в процессе обновлений, удаления или создания.
- `standby` — опциональный параметр, позволяющий заранее запускать дополнительные узлы в режиме ожидания.
- `priority` — целочисленный приоритет группы. При масштабировании `Cluster Autoscaler` сначала выбирает группы с большим значением `priority`. Используется для задания порядка масштабирования между несколькими группами узлов. Скоростью переключения на заказ узлов из групп с меньшим приоритетом можно управлять (подробнее — в разделе «FAQ»).

Пример конфигурации группы узлов с автомасштабированием:

```

spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1      # Минимальное количество узлов на зону.
    maxPerZone: 5      # Максимальное количество узлов на зону.
    maxUnavailablePerZone: 1 # Сколько узлов можно одновременно обновлять/удалять.
    zones:
      - nova
      - supernova
      - hypernova

```

Пример сценария автомасштабирования

Предположим, у вас есть следующая группа узлов:

```

spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: m4.large
    minPerZone: 1
    maxPerZone: 5
    zones:
      - nova
      - supernova
      - hypernova

```

Также есть Deployment с конфигурацией:

```

kind: Deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: app
          resources:
            requests:
              cpu: 1500m
              memory: 5Gi

```

В этом случае для запуска всех реплик потребуются 3 узла — по одному в каждую зону.

Теперь увеличим количество реплик до 5. В результате два пода окажутся в статусе `Pending`.

`Cluster Autoscaler`:

- отследит эту ситуацию;
- просчитает, сколько ресурсов не хватает;
- решит создать ещё два узла;
- передаст задание Machine Controller Manager;
- в облаке появятся 2 новые VM, которые автоматически подключатся к кластеру;
- поды будут размещены на новых узлах.

Выделение узлов под специфические нагрузки

 Запрещено использование домена `deckhouse.io` в ключах `labels` и `taints` у `NodeGroup`. Он зарезервирован для компонентов DKP. Следует отдавать предпочтение в пользу ключей `dedicated` или `dedicated.client.com`.

Для решений данной задачи существуют два механизма:

1. Установка лейблов в NodeGroup `spec.nodeTemplate.labels` для последующего использования их в Pod `spec.nodeSelector` или `spec.affinity.nodeAffinity`. Указывает, какие именно узлы будут выбраны планировщиком для запуска целевого приложения.
2. Установка ограничений в NodeGroup `spec.nodeTemplate.taints` с дальнейшим снятием их в Pod `spec.tolerations`. Запрещает исполнение не разрешенных явно приложений на этих узлах.

i

DKP по умолчанию поддерживает использование taint с ключом `dedicated`, поэтому рекомендуется применять этот ключ с любым значением для taints на ваших выделенных узлах.

Если требуется использовать другие ключи для taints (например, `dedicated.client.com`), необходимо добавить соответствующее значение ключа в параметр `modules.placement.customTolerationKeys`. Это обеспечит разрешение системным компонентам, таким как `cni-flannel`, использовать эти узлы.

Ускорение заказа узлов в облаке при горизонтальном масштабировании приложений

Для ускорения запуска новых реплик приложений при автоматическом горизонтальном масштабировании рекомендуется поддерживать в кластере определённое количество предварительно подготовленных (standby) узлов. Это позволяет быстро размещать новые поды приложений без ожидания создания и инициализации узлов. Следует учитывать, что наличие “запасных” узлов увеличивает расходы на инфраструктуру.

Необходимые настройки целевой NodeGroup будут следующие:

1. Указать абсолютное количество предварительно подготовленных узлов (или процент от максимального количества узлов в этой группе) в параметре `cloudInstances.standby`.
2. При наличии на узлах дополнительных служебных компонентов, не обслуживаемых Deckhouse (например, `DaemonSet filebeat`), задать их процентное потребление ресурсов узла можно в параметре `standbyHolder.overprovisioningRate`.
3. Для работы этой функции требуется, чтобы как минимум один узел из группы уже был запущен в кластере. Иными словами, либо должна быть доступна одна реплика приложения, либо количество узлов для этой группы `cloudInstances.minPerZone` должно быть 1.

Пример:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

Пример описания NodeGroup

Облачные узлы

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
  minPerZone: 1
  maxPerZone: 2
  classReference:
    kind: AWSInstanceClass
    name: test
  nodeTemplate:
    labels:
      tier: test

```

Конфигурация CloudEphemeral-узлов через NodeGroupConfiguration

Дополнительные настройки для облачных узлов можно задать с помощью объектов NodeGroupConfiguration. Они позволяют:

- изменять параметры ОС (например, `sysctl`);
- добавлять корневые сертификаты;
- настраивать доверие к собственным container registries и т.п.

NodeGroupConfiguration применяется к новым узлам при их создании, включая CloudEphemeral.

i NodeGroupConfiguration применяется только к узлам с указанным образом операционной системы (`bundle`). В качестве значения `bundle` можно указать как конкретное имя (например, `ubuntu-lts`, `centos-7`, `rocky-linux`), так и `*` — чтобы применить настройку ко всем образам ОС.

Примеры описания NodeGroupConfiguration

Задание параметра sysctl

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
    - "*"
  nodeGroups:
    - "*"
  content: |
    sysctl -w vm.max_map_count=262144

```

Добавление корневого сертификата в хост



- Данный пример приведен для ОС Ubuntu. Способ добавления сертификатов в хранилище может отличаться в зависимости от ОС. При адаптации скрипта под другую ОС измените параметры `bundles` и `content`.
- После добавления сертификата для его использования в `containerd` требуется перезапустить сервис `containerd` на узле.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
    - '*'
  bundles:
    - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAj4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )

    # bb-event          - Creating subscription for event function. More
information: http://www.bashbooster.net/#event
    ## ca-file-updated  - Event name
    ## update-certs     - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {      # Function with commands for adding a certificate to
the store
        update-ca-certificates
    }

    # bb-tmp-file - Creating temp file function. More information:
http://www.bashbooster.net/#tmp
    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    # bb-sync-file      - File synchronization function.
More information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - Destination file
    ## ${CERT_TMP_FILE}      - Source file
    ## ca-file-updated      - Name of event that will be called
if the file changes.

    bb-sync-file \
        "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
        ${CERT_TMP_FILE} \
        ca-file-updated
```

Добавление сертификата в ОС и containerd



Данный пример приведен для ОС Ubuntu.
Способ добавления сертификатов в хранилище может отличаться в зависимости от ОС.
При адаптации скрипта под другую ОС измените параметры `bundles` и `content`.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca-containerd.sh
spec:
  weight: 31
  nodeGroups:
    - '*'
  bundles:
    - 'ubuntu-lts'
  content: |-
    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAj4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )
    CONFIG_CONTENT=$(cat <<EOF
    [plugins]
    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
      ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
    )

    mkdir -p /etc/containerd/conf.d

    # bb-tmp-file - Create temp file function. More information:
    http://www.bashbooster.net/#tmp

    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    CONFIG_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CONFIG_CONTENT}" > "${CONFIG_TMP_FILE}"

    # bb-event - Creating subscription for event function. More
    information: http://www.bashbooster.net/#event
    ## ca-file-updated - Event name
    ## update-certs - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {
      # Function with commands for adding a certificate to
      the store
      update-ca-certificates # Restarting the containerd service is not required as
      this is done automatically in the script 032_configure_containerd.sh
    }

    # bb-sync-file - File synchronization function.
    More information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - Destination file
    ## ${CERT_TMP_FILE} - Source file
    ## ca-file-updated - Name of event that will be called
    if the file changes.
```

```
bb-sync-file \
  "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
  ${CERT_TMP_FILE} \
  ca-file-updated

bb-sync-file \
  "/etc/containerd/conf.d/${REGISTRY_URL}.toml" \
  ${CONFIG_TMP_FILE}
```

Обновление ядра на узлах

Для дистрибутивов, основанных на Debian

Создайте ресурс NodeGroupConfiguration, указав в переменной `desired_version` shell-скрипта (параметр `spec.content` ресурса) желаемую версию ядра:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

Для дистрибутивов, основанных на CentOS

Создайте ресурс NodeGroupConfiguration, указав в переменной `desired_version` shell-скрипта (параметр `spec.content` ресурса) желаемую версию ядра:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="3.10.0-1160.42.2.el7.x86_64"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-dnf-install "kernel-${desired_version}"

```

Добавление CloudPermanent-узлов в облачном кластере

Чтобы добавить узлы типа CloudPermanent в облачный кластер DKP:

1. Убедитесь, что включён модуль облачного провайдера. Например, `cloud-provider-yandex`, `cloud-provider-openstack`, `cloud-provider-aws` и др.

Это можно проверить с помощью команды:

```
d8 k -n d8-system get modules
```

Или посмотреть в веб-интерфейсе Deckhouse.

2. Создайте объект NodeGroup с типом `CloudPermanent`. Узлы типа `CloudPermanent` управляются через Terraform, встроенный в DKP. Конфигурация таких узлов находится в объекте `(Provider)ClusterConfiguration`. Редактировать его нужно с помощью утилиты `dhctl` в установочном контейнере. Пример:

```
nodeGroups:
- name: cloud-permanent
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: ubuntu-24-04-cloud-amd64
    rootDiskSize: 20
    mainNetwork: default
  volumeTypeMap:
    nova: ceph-ssd
```

3. Укажите параметры шаблона инстанса. Поля внутри `instanceClass` зависят от конкретного облачного провайдера. Ниже приведён пример для OpenStack:

- `flavorName` — тип инстанса (ресурсы: CPU, RAM);
- `imageName` — образ ОС;
- `rootDiskSize` — размер системного диска (в ГБ);
- `mainNetwork` — имя сети;
- при необходимости: диск `etcd`, зоны, `volume types` и т.д.

Для других облаков названия и структура параметров могут отличаться. Актуальные поля можно посмотреть в описании CRD или в документации по соответствующему облачному провайдеру.

4. Примените конфигурацию с помощью `dhctl converge`. После редактирования `(Provider)ClusterConfiguration` выполните:

```
dhctl converge \
  --ssh-host <IP мастер-узла> \
  --ssh-user <имя пользователя> \
  --ssh-agent-private-keys /tmp/.ssh/<ключ>
```

Эта команда:

- запустит Terraform;
- создаст нужные виртуальные машины;
- выполнит на них установку DKP (через `bootstrap.sh`);
- зарегистрирует узлы в кластере.

5. Готово — новые узлы появятся в кластере автоматически. Их можно увидеть выполнив команду:

```
d8 k get nodes
```

Также список новых узлов доступен в веб-интерфейсе Deckhouse.

Deckhouse Kubernetes Platform может работать поверх сервисов Managed Kubernetes (например, GKE и EKS). При этом модуль `node-manager` обеспечивает управление конфигурацией и автоматизацию действий с узлами, но возможности могут быть ограничены API соответствующего облачного провайдера.

Добавление CloudStatic узла в кластер

Добавление статического узла можно выполнить вручную или с помощью Cluster API Provider Static (CAPS).

Вручную

Чтобы добавить новый статический узел в кластер вручную, выполните следующие шаги:

1. Для CloudStatic-узлов в облачных провайдерах, перечисленных ниже, выполните описанные в документации шаги:
 - для AWS;
 - для GCP;
 - для YC.
2. Используйте существующий или создайте новый ресурс `NodeGroup`. Параметр `nodeType` в ресурсе `NodeGroup` для статических узлов должен быть `Static` или `CloudStatic`.
3. Получите код скрипта в кодировке Base64 для добавления и настройки узла.

Пример получения кода скрипта в кодировке Base64 для добавления узла в NodeGroup worker:

```
NODE_GROUP=worker
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for- $\{NODE\_GROUP\}$  -o json | jq
'.data."bootstrap.sh"' -r
```

4. Выполните предварительную настройку нового узла в соответствии с особенностями вашего окружения. Например:
 - добавьте необходимые точки монтирования в файл `/etc/fstab` (NFS, Ceph и т. д.);
 - установите необходимые пакеты;
 - настройте сетевую связность между новым узлом и остальными узлами кластера.
5. Зайдите на новый узел по SSH и выполните следующую команду, вставив полученную в п. 3 Base64-строку:

```
echo <Base64-КОД-СКРИПТА> | base64 -d | bash
```

С помощью Cluster API Provider Static

Простой пример добавления статического узла в кластер с помощью Cluster API Provider Static (CAPS):

1. Подготовьте необходимые ресурсы.
 - Выделите сервер (или виртуальную машину), настройте сетевую связность и т. п., при необходимости установите специфические пакеты ОС и добавьте точки монтирования которые потребуются на узле.
 - Создайте пользователя (в примере — `caps`) с возможностью выполнять `sudo`, выполнив **на сервере** следующую команду:

```
useradd -m -s /bin/bash caps
usermod -aG sudo caps
```

- Разрешите пользователю выполнять команды через `sudo` без пароля. Для этого **на сервере** внесите следующую строку в конфигурацию `sudo` (отредактировав файл `/etc/sudoers`, выполнив команду `sudo visudo` или другим способом):

```
caps ALL=(ALL) NOPASSWD: ALL
```

- Сгенерируйте **на сервере** пару SSH-ключей с пустой парольной фразой:

```
ssh-keygen -t ed25519 -f caps-id -C "" -N ""
```

Публичный и приватный ключи пользователя `caps` будут сохранены в файлах `caps-id.pub` и `caps-id` в текущей директории на сервере.

- Добавьте полученный публичный ключ в файл `/home/caps/.ssh/authorized_keys` пользователя `caps`, выполнив в директории с ключами **на сервере** следующие команды:

```
mkdir -p /home/caps/.ssh
cat caps-id.pub >> /home/caps/.ssh/authorized_keys
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps/
```

В операционных системах семейства Astra Linux, при использовании модуля мандатного контроля целостности Parsec, сконфигурируйте максимальный уровень целостности для пользователя `caps`:

```
pdpl-user -i 63 caps
```

2. Создайте в кластере ресурс SSHCredentials.

В директории с ключами пользователя **на сервере** выполните следующую команду для получения закрытого ключа в формате Base64:

```
base64 -w0 caps-id
```

На любом компьютере с `d8 k`, настроенным на управление кластером, создайте переменную окружения со значением закрытого ключа созданного пользователя в Base64, полученным на предыдущем шаге:

```
CAPS_PRIVATE_KEY_BASE64=<ЗАКРЫТЫЙ_КЛЮЧ_В_BASE64>
```

Выполните следующую команду для создания в кластере ресурса `SSHCredentials` (здесь и далее также используйте `d8 k`, настроенный на управление кластером):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: SSHCredentials
metadata:
  name: credentials
spec:
  user: caps
  privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
EOF
```

3. Создайте в кластере ресурс `StaticInstance`, указав IP-адрес сервера статического узла:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  # Укажите IP-адрес сервера статического узла.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

4. Создайте в кластере ресурс `NodeGroup`. Параметр `count` обозначает количество `staticInstances`, подпадающих под `labelSelector`, которые будут добавлены в кластер, в данном случае 1:

i Поле `labelSelector` в ресурсе `NodeGroup` является неизменным. Чтобы обновить `labelSelector`, нужно создать новую `NodeGroup` и перенести в неё статические узлы, изменив их лейблы (`labels`).

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

С помощью Cluster API Provider Static для нескольких групп узлов

Пример использования фильтров в `label selector` `StaticInstance`, для группировки статических узлов и использования их в разных `NodeGroup`. В примере используются две группы узлов (`front` и `worker`), предназначенные для разных задач, которые должны содержать разные по характеристикам узлы — два сервера для группы `front` и один для группы `worker`.

1. Подготовьте необходимые ресурсы (3 сервера или виртуальные машины) и создайте ресурс `SSHCredentials`, аналогично п.1 и п.2 примера.
2. Создайте в кластере два ресурса `NodeGroup`:

i Поле `labelSelector` в ресурсе `NodeGroup` является неизменным. Чтобы обновить `labelSelector`, нужно создать новую `NodeGroup` и перенести в неё статические узлы, изменив их лейблы.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: front
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

3. Создайте в кластере ресурсы `StaticInstance`, указав актуальные IP-адреса серверов:

```

apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF

```

Добавление master-узлов в облачном кластере

Чтобы добавить master-узлы в облачном кластере:

1. Убедитесь, что включён модуль `control-plane-manager`.
2. Откройте файл ClusterConfiguration (например, OpenStackClusterConfiguration).
3. Добавьте или отредактируйте секцию `masterNodeGroup`:

```

masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.medium
    imageName: ubuntu-24-04-cloud-amd64
    rootDiskSize: 20
    mainNetwork: default

```

4. Примените изменения с помощью `dhctl converge`:

```

dhctl converge \
  --ssh-host <IP мастер-узла> \
  --ssh-user <пользователь> \
  --ssh-agent-private-keys /tmp/.ssh/<ключ>

```


Использование NodeGroup с приоритетом

С помощью параметра `priority` кастомного ресурса NodeGroup можно задавать порядок заказа узлов в кластере. Например, можно сделать так, чтобы сначала заказывались узлы типа *spot-node*, а если они закончились — обычные узлы. Или чтобы при наличии ресурсов в облаке заказывались узлы большего размера, а при их исчерпании — узлы меньшего размера.

Пример создания двух NodeGroup с использованием узлов типа `spot-node`:

```
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral
```

В приведенном выше примере, `Cluster Autoscaler` сначала попытается заказать узел типа *spot-node*. Если в течение 15 минут его не получится добавить в кластер, NodeGroup `worker-spot` будет поставлен на паузу (на 20 минут) и `Cluster Autoscaler` начнет заказывать узлы из NodeGroup `worker`.

Если через 30 минут в кластере возникнет необходимость развернуть еще один узел, `Cluster Autoscaler` сначала попытается заказать узел из NodeGroup `worker-spot` и только потом — из NodeGroup `worker`.

После того как NodeGroup `worker-spot` достигнет своего максимума (5 узлов в примере выше), узлы будут заказываться из NodeGroup `worker`.

Шаблоны узлов (labels/taints) для NodeGroup `worker` и `worker-spot` должны быть одинаковыми, или как минимум подходить для той нагрузки, которая запускает процесс увеличения кластера.

Состояния группы узлов и их интерпретация

Ready — группа узлов содержит минимально необходимое число запланированных узлов с состоянием `Ready` для всех зон.

Пример 1. Группа узлов в состоянии `Ready`:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
    - status: "True"
      type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
    - status: "True"
      type: Ready

```

Пример 2. Группа узлов в состоянии **Not Ready**:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
    - status: "False"
      type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
    - status: "True"
      type: Ready

```

Updating — группа узлов содержит как минимум один узел, в котором присутствует аннотация с префиксом `update.node.deckhouse.io` (например, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval — группа узлов содержит как минимум один узел, в котором присутствует аннотация `update.node.deckhouse.io/disruption-required` и отсутствует аннотация `update.node.deckhouse.io/disruption-approved`.

Scaling — рассчитывается только для групп узлов с типом `CloudEphemeral`. Состояние `True` может быть в двух случаях:

1. Когда число узлов меньше *желаемого числа узлов в группе*, то есть когда нужно увеличить число узлов в группе.
2. Когда какой-то узел помечается к удалению или число узлов больше *желаемого числа узлов*, то есть когда нужно уменьшить число узлов в группе.

Желаемое число узлов — это сумма всех реплик, входящих в группу узлов.

Пример. Желаемое число узлов равно 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  ...
  desired: 2
  ...
```

Error — содержит последнюю ошибку, возникшую при создании узла в группе узлов.

Дополнительные ресурсы

- Документация модуля `cloud-provider-aws`
- Документация модуля `cloud-provider-gcp`
- Документация модуля `cloud-provider-openstack`
- Документация модуля `cloud-provider-yandex`
- Документация модуля `control-plane-manager`
- Документация модуля `node-manager`

Добавление и управление bare-metal узлами

Подробнее с подбором ресурсов для кластера на bare-metal можно ознакомиться в руководстве «Требования к кластеру на bare-metal».

Добавление узлов в bare-metal кластере

Ручной способ

1. Включите модуль `node-manager`.
2. Создайте объект NodeGroup с типом `Static`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

В спецификации этого ресурса укажите тип узлов `Static`. Для всех объектов NodeGroup в кластере автоматически создаётся скрипт `bootstrap.sh`, с помощью которого узлы добавляются в группы. Когда узлы добавляются вручную, необходимо скопировать этот скрипт на сервер и выполнить.

Скрипт можно получить в веб-интерфейсе Deckhouse на вкладке «Группы узлов → Скрипты» или командой `d8 k`:

```
d8 k -n d8-cloud-instance-manager get secrets manual-bootstrap-for-worker -ojsonpath="{.data.bootstrap\.sh}"
```

Скрипт нужно раскодировать из Base64, а затем выполнить от `root`.

3. Когда скрипт выполнится, сервер добавится в кластер в качестве узла той группы, для которой был использован скрипт.

Автоматический способ



Если вы ранее увеличивали количество master-узлов в кластере в NodeGroup `master` (параметр `spec.staticInstances.count`), перед добавлением в кластер узлов автоматическим способом убедитесь, что не произойдет их «перехват».

В DKP возможно автоматическое добавление физических (bare-metal) серверов в кластер без ручного запуска установочного скрипта на каждом узле. Для этого необходимо:

1. Подготовить сервер (ОС, сеть):
 - установить поддерживаемую ОС;
 - настроить сеть и убедиться, что сервер доступен по SSH;
 - создать системного пользователя (например, `ubuntu`), от имени которого будет выполняться подключение по SSH;
 - убедиться, что пользователь может выполнять команды через `sudo`.
2. Создать объект `SSHCredentials` с доступом к серверу. DKP использует объект `SSHCredentials` для подключения к серверам по SSH. В нём указывается:
 - приватный ключ, закодированный в формате Base64;
 - пользователь ОС;
 - порт SSH;
 - (опционально) пароль для `sudo`, если требуется.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: SSHCredentials
metadata:
  name: static-nodes
spec:
  privateSSHKey:
    LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KTUljRXZBSUJBREFOQmdrcWhraUc
    5dzBCQVFFRkFBT0NBZzhBTUljQkNnS0NBUEUUVB
  sshPort: 22
  sudoPassword: password
  user: ubuntu
```



Важно. Поле `privateSSHKey` должно содержать приватный SSH-ключ, закодированный в формате Base64. Приватный ключ должен соответствовать открытому ключу, добавленному в `~/.ssh/authorized_keys` на сервере.

Подробнее о защите конфиденциальных данных `SSHCredentials` — в FAQ модуля `node-manager`.

3. Создать объект `StaticInstance` для каждого сервера:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-0
  labels:
    static-node: auto
spec:
  address: 192.168.1.10
  credentialsRef:
    apiVersion: deckhouse.io/v1alpha1
    kind: SSHCredentials
    name: static-nodes
```

Под каждый сервер необходимо создавать отдельный ресурс `StaticInstance`, но можно

использовать одни и те же SSHCredentials для доступа на разные серверы.

Возможные состояния ресурсов StaticInstance:

- `Pending` — сервер ещё не настроен, в кластере отсутствует соответствующий узел;
- `Bootstrapping` — выполняется настройка сервера и подключение узла в кластер;
- `Running` — сервер успешно настроен, узел подключён к кластеру;
- `Cleaning` — выполняется очистка сервера и удаление узла из кластера.

Эти состояния отображают текущий этап управления узлом. CAPS автоматически переводит StaticInstance между этими состояниями в зависимости от необходимости добавить или удалить узел из группы.

4. Создать NodeGroup с описанием, как DKP будет использовать эти серверы:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 3
    labelSelector:
      matchLabels:
        static-node: auto
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
```

Здесь добавляются параметры, которые описывают использование StaticInstances:

- `count` указывает, сколько узлов будет добавлено в эту группу;
- в `labelSelector` прописываются правила для создания выборки узлов.

При использовании Cluster API Provider Static (CAPS) важно правильно задать параметры `nodeType: Static` и секцию `staticInstances` в объекте NodeGroup:

- Если параметр `labelSelector` не задан, CAPS будет использовать любые ресурсы StaticInstance, доступные в кластере.
- Один StaticInstance может использоваться в нескольких группах при совпадении лейблов.
- CAPS будет автоматически поддерживать количество узлов в группе, заданное в параметре `count`.
- При удалении узла CAPS выполнит его очистку и отключение, а соответствующий StaticInstance перейдёт в статус `Pending` и может быть использован повторно.

После создания группы узлов появится скрипт для добавления серверов в группу. DKP будет ожидать появления требуемого числа объектов StaticInstance, подходящих по лейблам. Как только такой объект появится, DKP использует указанный IP и параметры для подключения по SSH, выполнит скрипт `bootstrap.sh` и добавит сервер в группу.

Изменение конфигурации статического кластера

Настройки статического кластера хранятся в структуре StaticClusterConfiguration.

Чтобы изменить параметры статического кластера, выполните команду:

```
d8 system edit static-cluster-configuration
```

Перемещение статического узла между NodeGroup



В процессе переноса статических узлов между NodeGroup выполняется очистка и повторный bootstrap узла, объект `Node` пересоздаётся.

1. Создайте новый ресурс NodeGroup, например, с именем `front`, который будет управлять статическим узлом с лейблом `role: front`:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: front
```

- Измените лейбл `role` у существующего `StaticInstance` с `worker` на `front`. Это позволит новой `NodeGroup front` начать управлять этим узлом:

```
d8 k label staticinstance static-worker-1 role=front --overwrite
```

- Обновите ресурс `NodeGroup worker`, уменьшив значение параметра `count` с `1` до `0`:

```
d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --
type=merge
```

Ручная очистка статического узла

Для отключения узла из кластера и очистки сервера используйте скрипт `/var/lib/bashible/cleanup_static_node.sh`, который заранее размещён на каждом статическом узле.

Пример отключения узла кластера и очистки сервера:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-
doing
```

i Инструкция справедлива как для узла, настроенного вручную (с помощью бутстрап-скрипта), так и для узла, настроенного с помощью CAPS.

Пример описания NodeGroup

Пример описания NodeGroup для статических узлов

Для виртуальных машин на гипервизорах или физических серверов используйте статические узлы, указав `nodeType: Static` в `NodeGroup`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Узлы в такую группу добавляются вручную с помощью подготовленных скриптов или автоматически с помощью Cluster API Provider Static (CAPS).

Изменение CRI для NodeGroup

CRI (Container Runtime Interface) — стандартный интерфейс между kubelet и программой исполнения контейнеров (runtime).



Смена CRI возможна только между `Containerd` на `NotManaged` и обратно (параметр `cri.type`).

Для изменения CRI для NodeGroup, установите параметр `cri.type` в `Containerd` или в `NotManaged`.

Пример YAML-манифеста NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

Также эту операцию можно выполнить с помощью патча:

- Для `Containerd`:

```
d8 k patch nodegroup <имя NodeGroup> --type merge -p '{"spec":{"cri":
{"type":"Containerd"}}}'
```

- Для `NotManaged`:

```
d8 k patch nodegroup <имя NodeGroup> --type merge -p '{"spec":{"cri":
{"type":"NotManaged"}}}'
```



При изменении `cri.type` для NodeGroup, созданных с помощью `dhctl`, необходимо обновить это значение в `dhctl config edit provider-cluster-configuration` и настройках объекта NodeGroup.

После изменения CRI для NodeGroup модуль `node-manager` будет поочередно перезагружать узлы, применяя новый CRI. Обновление узла сопровождается простым (`disruption`). В зависимости от настройки `disruption` для NodeGroup, модуль `node-manager` либо автоматически выполнит обновление узлов, либо потребует подтверждения вручную.

Изменение NodeGroup у статического узла

Если узел находится под управлением CAPS, то изменить принадлежность к NodeGroup у такого узла **нельзя**. Единственный вариант — удалить `StaticInstance` и создать новый.

Чтобы перенести существующий статический узел, созданный вручную, из одной NodeGroup в другую, необходимо изменить у узла лейбл группы:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

Применение изменений потребует некоторого времени.

Изменение IP-адреса в StaticInstance

Изменить IP-адрес в ресурсе `StaticInstance` нельзя. Если в `StaticInstance` указан ошибочный адрес, то нужно удалить `StaticInstance` и создать новый.

Удаление StaticInstance

`StaticInstance`, находящийся в состоянии `Pending` можно удалять без ограничений.

Чтобы удалить `StaticInstance` находящийся в любом состоянии, отличном от `Pending` (`Running`, `Cleaning`, `Bootstrapping`), выполните следующие шаги:

1. Добавьте лейбл `"node.deckhouse.io/allow-bootstrap": "false"` в StaticInstance.

Пример команды для добавления лейбла:

```
d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
```

2. Дождитесь, пока StaticInstance перейдет в статус `Pending`.

Для проверки статуса StaticInstance используйте команду:

```
d8 k get staticinstances
```

3. Удалите StaticInstance.

Пример команды для удаления StaticInstance:

```
d8 k delete staticinstance d8cluster-worker
```

4. Уменьшите значение параметра `NodeGroup.spec.staticInstances.count` на 1.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `node-manager`

Пользовательские настройки на узлах

Для автоматизации действий на узлах группы предусмотрен ресурс `NodeGroupConfiguration`. С его помощью можно выполнять на узлах bash-скрипты, используя набор команд Bash Booster, а также применять шаблонизатор Go Template. Это удобно для автоматизации таких операций, как:

- Установка и настройки дополнительных пакетов ОС.

Примеры:

- установка kubectl-плагина;
- настройка containerd с поддержкой Nvidia GPU.

- Обновление ядра ОС на конкретную версию.

Примеры:

- обновление ядра Debian;
- обновление ядра CentOS.

- Изменение параметров ОС.

Примеры:

- настройка параметра sysctl;
- добавление корневого сертификата.

- Сбор информации на узле и выполнение других подобных действий.

Ресурс `NodeGroupConfiguration` позволяет указывать приоритет выполняемым скриптам, ограничивать их выполнение определенными группами узлов и типами ОС.

Код скрипта указывается в параметре `content` ресурса. При создании скрипта на узле содержимое параметра `content` проходит через шаблонизатор Go Template, который позволяет встроить дополнительный уровень логики при генерации скрипта. При прохождении через шаблонизатор становится доступным контекст с набором динамических переменных.

Переменные, которые доступны для использования в шаблонизаторе:

- `.cloudProvider` (для групп узлов с `nodeType` `CloudEphemeral` или `CloudPermanent`) — массив данных облачного провайдера.

Пример данных...


```
cloudProvider:
  instanceClassKind: OpenStackInstanceClass
  machineClassKind: OpenStackMachineClass
  openstack:
    connection:
      authURL: https://cloud.provider.com/v3/
      domainName: Default
      password: p@ssw0rd
      region: region2
      tenantName: mytenantname
      username: mytenantusername
    externalNetworkNames:
      - public
    instances:
      imageName: ubuntu-24-04-cloud-amd64
      mainNetwork: kube
      securityGroups:
        - kube
      sshKeyPairName: kube
    internalNetworkNames:
      - kube
    podNetworkMode: DirectRoutingWithPortSecurityEnabled
    region: region2
    type: openstack
    zones:
      - nova
```

- `.cri` — используемый CRI (с версии Deckhouse 1.49 используется только `Containerd`).
- `.kubernetesVersion` — используемая версия Kubernetes.
- `.nodeUsers` — массив данных о пользователях узла, добавленных через ресурс `NodeUser`.

Пример данных...

```
nodeUsers:
- name: user1
  spec:
    isSudoer: true
    nodeGroups:
      - '*'
    passwordHash: PASSWORD_HASH
    sshPublicKey: SSH_PUBLIC_KEY
    uid: 1050
```

- `.nodeGroup` — массив данных группы узлов.

Пример данных...

```
nodeGroup:
  cri:
    type: Containerd
  disruptions:
    approvalMode: Automatic
  kubelet:
    containerLogMaxFiles: 4
    containerLogMaxSize: 50Mi
    resourceReservation:
      mode: "Off"
  kubernetesVersion: "1.31"
  manualRolloutID: ""
  name: master
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
    taints:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
  nodeType: CloudPermanent
  updateEpoch: "1699879470"
```

Пример использования переменных в шаблонизаторе:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

Пример использования команд Bash Booster:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Мониторинг выполнения скриптов

Применение `NodeGroupConfiguration` запускает обновление **конфигурации `bashible` группы узлов** в указанных группах. DKP автоматически обнаруживает изменение этой конфигурации, затем выставляет поле `UPTODATE` (в выводе команды `d8 k get nodegroup`) в `0`. Поле `UPTODATE` показывает, сколько узлов уже приведены к **целевой конфигурации `bashible` группы узлов**; значение `0` означает, что пока ни один узел не приведён к целевой конфигурации `bashible` (т.е. обновление ещё не применено ни на одном узле).

DKP управляет запуском обновления на узлах через аннотацию `update.node.deckhouse.io/approved` на объекте `Node`. По умолчанию обновление выполняется одновременно на одном узле из каждой группы. Параллельность обновления задаётся параметром `maxConcurrent` в конфигурации группы узлов. Когда DKP выбирает узел для обновления (с учётом очереди и `maxConcurrent`), на него проставляется аннотация `update.node.deckhouse.io/approved`, после чего сервис `bashible` на этом узле начинает применять **целевую конфигурацию `bashible` группы узлов**.

Проставленные DKP аннотации узлов на запуск обновления можно посмотреть с помощью команды:

```
d8 k get nodes -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") |
.metadata.name' -r
```

Ход выполнения скриптов можно увидеть на узле в журнале сервиса `bashible` с помощью

команды:

```
journalctl -fu bashible.service
```

Сами скрипты находятся на узле в директории `/var/lib/bashible/bundle_steps/`.

Механизм повторного запуска скриптов

Сервис принимает решение о повторном запуске скриптов путем сравнения единой контрольной суммы всех файлов, расположенной по пути `/var/lib/bashible/configuration_checksum` с контрольной суммой размещенной в кластере Kubernetes в секрете `configuration-checksums` пространства имён `d8-cloud-instance-manager`.

Проверить контрольную сумму можно следующей командой:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o yaml
```

Сравнение контрольных сумм сервис совершает каждую минуту.

Контрольная сумма в кластере изменяется раз в 4 часа, тем самым повторно запуская скрипты на всех узлах.

Принудительный запуск скриптов `bashible` на узле можно выполнить, удалив файл контрольной суммы с помощью следующей команды:

```
rm /var/lib/bashible/configuration_checksum
```

Особенности написания скриптов

При написании скриптов важно учитывать следующие особенности их использования в DKP:

1. Скрипты в DKP выполняются раз в 4 часа или на основании внешних триггеров. Поэтому важно писать скрипты таким образом, чтобы они производили предварительную проверку необходимости внесения изменений, чтобы избежать повторяющихся/ненужных действий при каждом запуске.
2. Существуют заранее подготовленные скрипты, выполняющие различные действия, в том числе установку и настройку сервисов. Важно учитывать это при выборе приоритета пользовательских скриптов. Например, если пользовательский скрипт выполняет перезапуск сервиса, он должен запускаться после скрипта, отвечающего за установку этого сервиса. В противном случае пользовательский скрипт не сможет выполниться при первичном развёртывании узла (так как сервис ещё не будет установлен).

Полезные особенности некоторых скриптов:

- `032_configure_containerd.sh` — объединяет конфигурационные файлы `containerd` и **перезапускает** сервис. Размещение конфигурационных файлов зависит от CRI на узлах кластера:
 - `/etc/containerd/conf.d/*.toml` — для `containerd v1`;
 - `/etc/containerd/conf2.d/*.toml` — для `containerd v2`.

Следует учитывать, что директории с конфигурационными файлами не создаются автоматически. Файлы в них нужно добавлять в скриптах с приоритетом менее `32`.

Как использовать containerd с поддержкой Nvidia GPU

Создайте отдельную NodeGroup для GPU-узлов:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu
spec:
  chaos:
    mode: Disabled
  disruptions:
    approvalMode: Automatic
  nodeType: CloudStatic

```

Далее создайте ресурс NodeGroupConfiguration для NodeGroup `gpu` для конфигурации containerd:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    # http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/nvidia_gpu.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".containerd]
          default_runtime_name = "nvidia"
        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]
          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
            [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]
              privileged_without_host_devices = false
              runtime_engine = ""
              runtime_root = ""
              runtime_type = "io.containerd.runc.v2"

      [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]
        BinaryName = "/usr/bin/nvidia-container-runtime"
        SystemdCgroup = false

    EOF
  nodeGroups:
    - gpu
  weight: 31

```

Добавьте ресурс NodeGroupConfiguration для установки драйверов Nvidia для NodeGroup `gpu`.

Ubuntu

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-cuda.sh
spec:
  bundles:
  - ubuntu-lts
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    if [ ! -f "/etc/apt/sources.list.d/nvidia-container-toolkit.list" ]; then
      distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
      curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-key
add -
      curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.list | sudo tee
/etc/apt/sources.list.d/nvidia-container-toolkit.list
    fi
    bb-apt-install nvidia-container-toolkit nvidia-driver-535-server
    nvidia-ctk config --set nvidia-container-runtime.log-level=error --in-place
  nodeGroups:
  - gpu
  weight: 30

```

CentOS

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-cuda.sh
spec:
  bundles:
  - centos
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    # http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    if [ ! -f "/etc/yum.repos.d/nvidia-container-toolkit.repo" ]; then
      distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
      curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.repo | sudo tee /etc/yum.repos.d/nvidia-
container-toolkit.repo
    fi
    bb-dnf-install nvidia-container-toolkit nvidia-driver
    nvidia-ctk config --set nvidia-container-runtime.log-level=error --in-place
  nodeGroups:
  - gpu
  weight: 30

```

После того как конфигурации будут применены, проведите бутстрап и перезагрузите узлы, чтобы применить настройки и установить драйвера.

Особенности использования MIG в кластере из одного узла

i Общая информация о подготовке GPU-узлов, установке драйверов и проверке работы GPU приведена в разделе «Как работать с GPU-узлами».

В одноузловом кластере при использовании стратегии **MIG** после перезагрузки узла требуется вручную повторно применить MIG-конфигурацию.

Это ограничение относится к текущей реализации MIG, в которой используется компонент **nvidia-mig-manager**.

После каждой перезагрузки узла выполните следующие шаги:

1. Пометьте узел служебной аннотацией:

```
d8 k annotate node <node_name> update.node.deckhouse.io/drained=""
```

2. Перезапустите компонент **nvidia-mig-manager**:

```
d8 k -n d8-nvidia-gpu delete pod -l component=nvidia-mig-manager
```

3. Дождитесь повторного применения MIG-конфигурации по логам **nvidia-mig-manager**:

```
d8 k -n d8-nvidia-gpu logs -f -l component=nvidia-mig-manager
```

Успешное применение конфигурации подтверждается записью вида:

```
Successfully updated to MIG config: <partedConfig>
```

где `<partedConfig>` — значение параметра `partedConfig` в NodeGroup (например, `all-1g.10gb`).

- После успешного применения MIG-конфигурации верните узел в рабочее состояние:

```
d8 k uncordon <node_name>
```

Проверка успешности установки

Создайте в кластере Job:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: nvidia-cuda-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      containers:
        - name: nvidia-cuda-test
          image: nvidia/cuda:11.6.2-base-ubuntu20.04
          imagePullPolicy: "IfNotPresent"
          command:
            - nvidia-smi
```

Проверьте логи командой:

```
d8 k logs job/nvidia-cuda-test
```

Пример вывода:

```
Tue Jan 24 11:36:18 2023
+-----+
| NVIDIA-SMI 525.60.13      Driver Version: 525.60.13      CUDA Version: 12.0      |
+-----+-----+-----+-----+-----+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M.         |
+-----+-----+-----+-----+-----+-----+
|   0   Tesla T4              Off      | 00000000:8B:00:00 Off |                    0 |
| N/A   45C    P0       25W /  70W |  0MiB / 15360MiB |      0%      Default |
|                                           | N/A            |
+-----+-----+-----+-----+-----+

+-----+
| Processes:
|  GPU   GI    CI          PID    Type   Process name                      GPU Memory
|          ID    ID                                   |          Usage          |
+-----+-----+-----+-----+-----+
| No running processes found
+-----+
```

Создайте в кластере Job:

```

apiVersion: batch/v1
kind: Job
metadata:
  name: gpu-operator-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      containers:
        - name: gpu-operator-test
          image: nvidia/samples:vectoradd-cuda10.2
          imagePullPolicy: "IfNotPresent"

```

Проверьте логи командой:

```
d8 k logs job/gpu-operator-test
```

Пример вывода:

```

[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done

```

Как развернуть кастомный конфигурационный файл containerd



Добавление кастомных настроек вызывает перезапуск сервиса `containerd`.

Bashible на узлах объединяет конфигурацию `containerd` для Deckhouse с конфигурацией из файла:

- `/etc/containerd/conf.d/*.toml` — если в качестве CRI на узлах кластера используется `containerd v1`.
- `/etc/containerd/conf2.d/*.toml` — если в качестве CRI на узлах кластера используется `containerd v2`.



Вы можете переопределять значения параметров, которые заданы в файле `/etc/containerd/deckhouse.toml`. При этом корректную работу таких изменений необходимо обеспечить самостоятельно. Рекомендуется **не изменять** конфигурацию на управляющих (master) узлах (NodeGroup master).

- Для `containerd v1`
- Для `containerd v2`


```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-
shm.options]
      BinaryName = ""
      SystemdCgroup = true
      ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31

```

Добавление конфигурации для дополнительного registry

В containerd существует два способа описания конфигурации registry: **устаревший** и **актуальный**.

Для проверки наличия **устаревшего** способа конфигурации выполните на узлах кластера следующие команды:

```

cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.configs'

```

Пример вывода:

```

[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
[plugins."io.containerd.grpc.v1.cri".registry.configs]
  [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]

```

Для проверки наличия **актуального** способа конфигурации выполните на узлах кластера следующую команду:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'
```

Пример вывода:

```
# config_path = "/etc/containerd/registry.d"
```

Устаревший способ добавления конфигурации для дополнительного registry



Этот формат конфигурации containerd устарел (deprecated).



Используется в containerd v1, если DKP не управляется с помощью модуля registry.

Конфигурация описывается в основном конфигурационном файле containerd `/etc/containerd/config.toml`.

Пользовательская конфигурация добавляется через механизм `toml merge`. Конфигурационные файлы из директории `/etc/containerd/conf.d` объединяются с основным файлом `/etc/containerd/config.toml`. Применение merge происходит на этапе выполнения скрипта `032_configure_containerd.sh`, поэтому соответствующие файлы должны быть добавлены заранее.

Пример конфигурационного файла для директории `/etc/containerd/conf.d/`:

```
[plugins]
[plugins."io.containerd.grpc.v1.cri"]
[plugins."io.containerd.grpc.v1.cri".registry]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
  endpoint = ["https://${REGISTRY_URL}"]
[plugins."io.containerd.grpc.v1.cri".registry.configs]
[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
  auth = "${BASE_64_AUTH}"
  username = "${USERNAME}"
  password = "${PASSWORD}"
[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
  ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
  insecure_skip_verify = true
```



Добавление кастомных настроек через механизм `toml merge` вызывает перезапуск сервиса containerd.

Добавление авторизации в дополнительный registry (устаревший способ)

Пример добавления авторизации в дополнительный registry при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
      username = "username"
      password = "password"
      # OR
      auth = "dXNlc m5hbWU6cGFzc3dvcmQ="
    EOF

```

Настройка сертификата для дополнительного registry (устаревший способ)

Пример настройки сертификата для дополнительного registry при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"

    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.cert" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
      ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.cert"
    EOF

```

i Помимо containerd, сертификат можно добавить в операционную систему.

Добавление TLS skip verify (устаревший способ)

Пример добавления TLS skip verify при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
      insecure_skip_verify = true
    EOF

```

После применения конфигурационного файла проверьте доступ к registry с узлов, используя команду:

```

# Через cri-интерфейс
crictl pull private.registry.example/image/repo:tag

```

Настройка зеркала для доступа к публичным registries (устаревший способ)

Пример настройки зеркала к публичным registries при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
mkdir -p /etc/containerd/conf.d
bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
[plugins]
[plugins."io.containerd.grpc.v1.cri"]
[plugins."io.containerd.grpc.v1.cri".registry]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
  endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
  endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]

EOF

```

Новый способ добавления конфигурации для дополнительного registry

i

Используется в containerd v2.

Используется в containerd v1, если управление осуществляется через модуль `registry` (например, в режиме `Direct`).

Конфигурация описывается в каталоге `/etc/containerd/registry.d` и задаётся через создание подкаталогов с именами, соответствующими адресу registry:

```

/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.ru
    ├── ca.crt
    └── hosts.toml

```

Пример содержимого файла `hosts.toml`:

```
[host]
# Mirror 1.
[host."https://${REGISTRY_URL_1}"]
  capabilities = ["pull", "resolve"]
  ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

[host."https://${REGISTRY_URL_1}.auth"]
  username = "${USERNAME}"
  password = "${PASSWORD}"

# Mirror 2.
[host."http://${REGISTRY_URL_2}"]
  capabilities = ["pull", "resolve"]
  skip_verify = true
```

i Изменения конфигураций не приводят к перезапуску сервиса containerd.

Добавление авторизации в дополнительный registry (актуальный способ)

Пример добавления авторизации в дополнительный registry при использовании **актуального** способа конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Шаг может быть любой, так как не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    # http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
    [host."https://${REGISTRY_URL}"]
      capabilities = ["pull", "resolve"]
    [host."https://${REGISTRY_URL}.auth"]
      username = "username"
      password = "password"
    EOF
```


Настройка сертификата для дополнительного registry (актуальный способ)

Пример настройки сертификата для дополнительного registry при использовании **актуального** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # Шаг может быть любой, так как не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
    [host."https://${REGISTRY_URL}"]
    capabilities = ["pull", "resolve"]
    ca = ["/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt"]
    EOF

```



Помимо containerd, сертификат можно добавить в операционную систему.

Как добавить TLS skip verify (актуальный способ)?

Пример добавления TLS skip verify при использовании **актуального** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # Шаг может быть любой, так как не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        skip_verify = true
    EOF

```

После применения конфигурационного файла проверьте доступ к registry с узлов, используя команды:

```

# Через cri интерфейс.
crictl pull private.registry.example/image/repo:tag

# Через ctr с указанием директории с конфигурациями.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/image/repo:tag

# Через ctr для http репозитория.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/image/repo:tag

```

Настройка зеркала для доступа к публичным registries (актуальный способ)

Пример настройки зеркала к публичным registries при использовании **актуального** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF

```

Как автоматически проставить на узел кастомные лейблы

1. На узле создайте каталог `/var/lib/node_labels`.
2. Создайте в нём файл или файлы, содержащие необходимые лейблы. Количество файлов может быть любым, как и вложенность подкаталогов, их содержащих.
3. Добавьте в файлы нужные лейблы в формате `key=value`. Например:

```
example-label=test
```

4. Сохраните файлы.

При добавлении узла в кластер указанные в файлах лейблы будут автоматически проставлены на узел.



Обратите внимание, что добавить таким образом лейблы, используемые в DKP, невозможно. Работать такой метод будет только с кастомными лейблами, не пересекающимися с зарезервированными для Deckhouse.

Дополнительные ресурсы

- Параметры модуля `deckhouse`
- Документация модуля `node-manager`
- Документация модуля `registry`

Миграция container runtime на containerd v2

Вы можете настроить containerd v2 как основной container runtime на уровне всего кластера или для отдельных групп узлов. Этот вариант позволяет использовать cgroups v2, обеспечивает лучшую безопасность и более гибкое управление ресурсами.

Требования

Миграция на containerd v2 возможна при выполнении следующих условий:

- Узлы соответствуют требованиям, описанным в общих параметрах кластера.
- Версия ядра узла не входит в диапазон 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS).
- На сервере нет кастомных конфигураций в `/etc/containerd/conf.d` (пример кастомной конфигурации).



На узлах с Ubuntu в AWS модуль ядра `erofs`, необходимый для containerd v2, поставляется отдельным пакетом `linux-modules-extra`. DKP устанавливает его автоматически, но для части ядер, преимущественно устаревших, этот пакет больше не публикуется в APT-репозитории. В этом случае установка пропускается, в логах узла появляется предупреждение, узел получает лейбл `node.deckhouse.io/containerd-v2-unsupported`, и по нему срабатывает алерт `D8NodeContainerdV2NotSupported`.

Проблема подтверждена на ядрах `5.15.0-1028-aws`, `6.8.0-1024-aws` и `6.8.0-1029-aws`. Чтобы мигрировать такие узлы, переключите группу узлов на более свежий AMI или ядро, в котором есть `erofs`.

Перед миграцией проверьте, есть ли в кластере такие узлы: сверьтесь с алертами и списком узлов с этим лейблом (команды приведены ниже).

При несоответствии одному из требований, описанных в общих параметрах кластера, Deckhouse Kubernetes Platform добавляет на узел лейбл `node.deckhouse.io/containerd-v2-unsupported`. Если на узле есть кастомные конфигурации в директориях `/etc/containerd/conf.d/` (если на узлах кластера используется CRI containerd v1) или `/etc/containerd/conf2.d/` (если на узлах кластера используется CRI containerd v2), на него добавляется лейбл `node.deckhouse.io/containerd-config=custom`.

При наличии одного из этих лейблов смена параметра `spec.cri.type` для группы узлов будет недоступна. Узлы, которые не подходят под условия миграции можно посмотреть с помощью команд:

```
d8 k get node -l node.deckhouse.io/containerd-v2-unsupported
d8 k get node -l node.deckhouse.io/containerd-config=custom
```

Также администратор может проверить конкретный узел на соответствие требованиям с помощью команд:

```
uname -r | cut -d- -f1
stat -f -c %T /sys/fs/cgroup
systemctl --version | awk 'NR==1{print $2}'
modprobe -qn erofs && echo "TRUE" || echo "FALSE"
ls -l /etc/containerd/conf.d
```

Как включить containerd v2

i В процессе миграции директория `/var/lib/containerd` будет очищена, что приведет к повторному скачиванию образов всех подов, и узел перезагрузится.

Также обратите внимание, что обновление узла приводит к прерыванию его работы (disruptive-обновлению). Если для группы узлов установлено значение `spec.disruptions.approvalMode: Manual`, подтвердите обновление вручную на соответствующих узлах с помощью аннотации: `d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=`.

Включение containerd v2 возможно двумя способами:

1. **Для всего кластера.** Укажите значение `ContainerdV2` в параметре `defaultCRI` ресурса `ClusterConfiguration`. Это значение будет применяться ко всем `NodeGroup`, в которых явно не указан `spec.cri.type`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
...
defaultCRI: ContainerdV2
```

2. **Для конкретной группы узлов.** Укажите `ContainerdV2` в параметре `spec.cri.type` в объекте `NodeGroup`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cri:
    type: ContainerdV2
```

При переходе на containerd v2 Deckhouse Kubernetes Platform начнет поочерёдное обновление узлов.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `node-manager`

Масштабирование приложений и управление ресурсами

Масштабирование приложений и управление размещением подов в Deckhouse Kubernetes Platform позволяет динамически адаптировать ресурсы кластера под текущую нагрузку, обеспечивая высокую доступность сервисов и эффективное использование ресурсов.

Deckhouse поддерживает все основные механизмы масштабирования и распределения нагрузок в Kubernetes:

- Горизонтальное масштабирование (HPA) — автоматическое изменение количества реплик подов на основе метрик потребления ресурсов или внешних показателей.
- Вертикальное масштабирование (VPA) — автоматическая настройка запрашиваемых ресурсов (CPU и памяти) для контейнеров в зависимости от их фактического потребления.
- Масштабирование по метрикам — использование произвольных метрик для гибкого управления масштабированием приложений через Prometheus.
- Перераспределение подов (Descheduler) — автоматическое вытеснение подов для оптимизации размещения рабочих нагрузок в кластере.
- Классы приоритета (Priority Classes) — управление очередностью вытеснения подов при нехватке ресурсов на основе их важности.
- Планировщик (Scheduler) — настройка правил и логики выбора узлов для размещения подов.

Deckhouse Kubernetes Platform обеспечивает автоматическое масштабирование приложений и

эффективное управление ресурсами для стабильной и предсказуемой работы кластера.

В следующих разделах подробно описаны возможности масштабирования, примеры конфигураций и рекомендации по их эффективному использованию.

Масштабирование по метрикам

Масштабирование по метрикам

Масштабирование по метрикам — это процесс автоматического или ручного изменения ресурсов (например, количества реплик подов, выделенных CPU/памяти) на основе определённых метрик. Метрики могут быть как по использованию CPU или памяти, так и по количеству запросов в секунду или размер очереди сообщений.

Масштабирование может выполняться на основе как базовых, так и произвольных метрик мониторинга. Например, масштабирование может осуществляться по следующим метрикам:

- CPU (пода) — текущее использование процессора.
- Память (пода) — текущее использование оперативной памяти.
- RPS (Ingress) — количество запросов в секунду за 1, 5, 15 минут (`rps 1m`, `rps 5m`, `rps 15m`).
- Среднее потребление CPU (пода) — за 1, 5, 15 минут (`cpu 1m`, `cpu 5m`, `cpu 15m`).
- Среднее потребление памяти (пода) — за 1, 5, 15 минут (`memory 1m`, `memory 5m`, `memory 15m`).

Настройка

Для упрощения настройки масштабирования по метрикам, DKP предлагает использовать специальные ресурсы (`Cluster*Metric` и `*Metric`). Также, можно настроить масштабирование по собственным правилам расчета метрик.

Для настройки масштабирования по метрикам, выполняйте следующие действия:

1. Включите модуль `prometheus-metrics-adapter`. Он позволяет создавать кастомные ресурсы, описывающие правила расчета метрики для масштабирования в виде PromQL-запроса.

Включить модуль `prometheus-metrics-adapter` можно в веб-интерфейсе Deckhouse, или с помощью следующей команды:

```
d8 system module enable prometheus-metrics-adapter
```

2. Определите PromQL-запрос, который будет описывать нужную метрику (например, скорость запросов к приложению или загрузку CPU за определённое время). Этот запрос регистрируется в кластере как метрика для HPA.
3. Создайте ресурс `HorizontalPodAutoscaler`, указав в нём имя метрики и пороговое значение. Kubernetes будет опрашивать результат вашего PromQL-запроса и сравнивать с заданным порогом, добавляя или убавляя реплики (читайте в документации подробнее про настройку горизонтального масштабирования).

Пример запроса PromQL:

```
query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>) OR on() vector(0)
```

В данном примере:

- `ingress_nginx_detail_requests_total` — базовая метрика (счётчик запросов, например от контроллера Ingress NGINX);
- `rate(...[2m])` — вычисляет скорость увеличения счётчика за последние 2 минуты (requests per second);
- `sum(...) by (<<.GroupBy>>)` — суммирует результат по некоторому набору лейблов (например, по имени Ingress, пространству имён и т. д.);
- `OR on() vector(0)` — значение для PromQL, позволяющий вернуть «0», когда нет данных (иначе результат мог бы быть пустым и не учитываться).

Таким образом, этот PromQL-запрос даёт «число текущих запросов в секунду» по указанным лейблам (Ingress, namespace и т. д.).

Когда у администратора есть метрика, описанная через PromQL-запрос (например, в объекте `IngressMetric` или аналогичном механизме), её можно указать в `HorizontalPodAutoscaler` (HPA) например так:

```

apiVersion: deckhouse.io/v1beta1
kind: IngressMetric
metadata:
  name: mymetric
  namespace: mynamespace
spec:
  query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by
(<<.GroupBy>>) OR on() vector(0)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  # Метрики, используемые для масштабирования.
  # Пример использования метрик.
  metrics:
  - type: Object
    object:
      # Объект, который обладает метриками в Prometheus.
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: myingress
      metric:
        # Метрика, зарегистрированная с помощью custom resource IngressMetric или
ClusterIngressMetric.
        # Можно использовать rps_1m, rps_5m или rps_15m которые поставляются с модулем
prometheus-metrics-adapter.
        name: mymetric
      target:
        # Для метрик типа Object можно использовать `Value` или `AverageValue`.
        type: AverageValue
        # Масштабирование происходит, если среднее значение метрики для всех подов в
Deployment сильно отличается от 10.
        averageValue: 10

```

i При превышении (или падении ниже) заданного порога `averageValue: 10` система изменит число реплик `myapp` в диапазоне от 1 до 2

Настройка метрик через ресурсы модуля observability

Чтобы настроить собственные правила расчёта метрик:

1. Убедитесь, что включён модуль `observability`.
2. Создайте ресурс `ClusterObservabilityMetricsRulesGroup` или `ObservabilityMetricsRulesGroup`:
 - `ClusterObservabilityMetricsRulesGroup` — предназначен для cluster-wide-правил, применяемых к системным неймспейсам;
 - `ObservabilityMetricsRulesGroup` — предназначен для правил, работающих в рамках конкретного неймспейса.

Пример конфигурации:

```
apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: custom-cluster-metrics-mymetric
spec:
  interval: 1m
  rules:
    - record: kube_adapter_metric_ingress_bytes_sent
      expr: |
        sum(ingress_nginx_detail_sent_bytes_sum) by (namespace, ingress)
```

Работа с нестабильными метриками

При работе с нестабильными метриками (например, если метрика колеблется и вызывает избыточное масштабирование) рекомендуется:

- Использовать агрегирующие функции PromQL. Например, `avg_over_time()` усредняет значение метрики за заданный промежуток времени, что помогает избежать резких скачков:

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
    (avg_over_time(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"}[5m])) by (<<.GroupBy>>)
```

- Настроить стабилизацию в поведении автоскейлера. Время стабилизации масштабирования можно увеличить, чтобы решения о добавлении или удалении реплик принимались на основе более устойчивых данных.

Получение значений метрик

Для получения списка метрик используйте команду:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/
```

Для получения значений метрик, привязанных к объектам используйте команду:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/services/*/my-service-metric
```

Для получения значений метрик, созданных через `NamespaceMetric` используйте команду:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/metrics/my-ns-metric
```

Для получения внешних метрик используйте команду:

```
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1
```

Настройка автомасштабирования в веб-интерфейсе Deckhouse

В Deckhouse Kubernetes Platform можно настраивать параметры автомасштабирования узлов через веб-интерфейс Deckhouse. Это позволяет динамически изменять количество узлов в зависимости от нагрузки.

Чтобы настроить автомасштабирование в веб-интерфейсе:

- Откройте веб-интерфейс Deckhouse;
- В левом меню перейдите в раздел «Узлы» → «Группы узлов»;
- Выберите нужную группу узлов;
- В разделе «Параметры автомасштабирования» отобразятся текущие настройки.

В веб-интерфейсе Deckhouse доступны следующие настройки:

- Параметр «Узлов на зону» – задаёт минимальное и максимальное количество узлов, которые могут быть запущены в одной зоне.
- Параметр «Необходимо» – текущее количество узлов, необходимых для работы.
- Параметр «Заказано» – текущее количество узлов, запланированное для запуска.
- Параметр «Резерв» – дополнительный запас узлов.

Чтобы изменить параметры автомасштабирования:

- Нажмите на значок редактирования рядом с параметром «Узлов на зону».
- В появившемся окне укажите минимальное и максимальное количество узлов.
- Нажмите «Применить», чтобы сохранить изменения, или «Отменить», чтобы сбросить редактирование.

При нажатии «Создать на основании» в веб-интерфейсе открывается форма настройки параметров автомасштабирования:

1. Основные параметры:

- Зоны – выбор зон, в которых будут создаваться машины. Позволяет распределять узлы между разными зонами доступности. Значение по умолчанию зависит от выбранного облачного провайдера и обычно соответствует всем зонам используемого региона
- Приоритет масштабирования – используется, если в кластере есть несколько групп узлов с одинаковым приоритетом. Если приоритет не указан, то система случайным образом выбирает группу для добавления новых узлов.
- Количество одновременно создаваемых машин при масштабировании вверх – определяет, сколько машин может быть добавлено одновременно при масштабировании вверх. Например, если значение 1, то узлы добавляются по одному.
- Недоступное количество инстансов при RollingUpdate – количество узлов, которые могут быть недоступны во время обновления (RollingUpdate). Значение 0 означает, что обновление будет проходить поочерёдно, без временного снижения числа работающих узлов.

2. Резервные узлы и ресурсы:

- Число резервных узлов от общего количества узлов – можно указать конкретное число узлов (например, 2-6) или процент от общего числа узлов (например, 15%). Резервный узел — это узел кластера, на котором резервируются ресурсы, доступные в любой момент для масштабирования. Наличие такого узла позволяет автоскейлеру кластера не ждать инициализации узла, а сразу размещать на нем нагрузку. Это позволяет ускорить масштабирование вверх на несколько минут.
- Ресурсы, занимаемые на резервных узлах, % – процент ресурсов, который выделяется на резервных узлах. Допустимые значения: от 1% до 80%. Например, если указать 50%, то половина ресурсов узла будет зарезервирована.

3. Дополнительные параметры:

- Quick Shutdown (Быстрое выключение) – включает ускоренное отключение узлов, снижая время ожидания drain'a до 5 минут.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `observability`
- Документация модуля `prometheus-metrics-adapter`

Горизонтальное масштабирование

Как работает горизонтальное масштабирование (HPA)

Horizontal Pod Autoscaler (HPA) — это механизм автоматического изменения (вверх или вниз) количества реплик подов (в Deployments или StatefulSets) на основе метрик, полученных через Kubernetes API. HPA следит за уровнем нагрузки на приложение, проверяя актуальные метрики (например, CPU, память или дополнительные из Prometheus), и при необходимости изменяет количество реплик для поддержания заданного уровня производительности или экономии ресурсов.

Доступные типы метрик для HPA

Горизонтальное масштабирование в DKP может выполняться по любым доступным метрикам, например:

1. По потреблению CPU и памяти подов.
 - Настраивается с помощью ресурса `HorizontalPodAutoscaler`. Например, можно указать метрику типа `Resource` с `averageUtilization = 70` для CPU, чтобы при среднем использовании 70% масштабировать приложение вверх.
2. По метрикам объектов DKP (Ingress, Service) или по метрикам самих подов (сумма или среднее по всем подам одного контроллера).
 - Позволяют масштабироваться на основе метрик, привязанных к объектам DKP (например, Ingress, Service), или рассчитанных на под (сумма или среднее по всем подам контроллера). Используются ресурсы `ServiceMetric` для сервисов и `IngressMetric` для Ingress.
3. По любым другим метрикам, включая внешние данные (метрики Amazon SQS, облачных балансировщиков, SaaS-сервисов и т. п.).
 - Используются, когда источник метрик находится вне кластера. Правила расчёта метрик настраиваются с помощью ресурсов `ClusterObservabilityMetricsRulesGroup` (для cluster-wide-метрик) и `ObservabilityMetricsRulesGroup` (для namespaced-метрик) и требуют включения модуля `observability`.

Рекомендации HPA

При колебаниях метрик оберните метрику в агрегирующую функцию (например, `avg over time()`) или увеличьте время стабилизации (`spec.behavior.scaleUp.stabilizationWindowSeconds`), чтобы избежать резкого увеличения количества подов.

Ограничения HPA

1. По умолчанию HPA по-разному обрабатывает масштабирование вверх и вниз:
 - Масштабирование вверх:
 - Происходит незамедлительно (`spec.behavior.scaleUp.stabilizationWindowSeconds = 0`).
 - Ограничение — за 15 секунд число подов может удвоиться. Если подов было меньше 4, то добавятся 4 новых пода.
 - Масштабирование вниз:
 - Происходит в течение 5 минут (`spec.behavior.scaleDown.stabilizationWindowSeconds = 300`).
 - Собираются «предложения» о новом количестве реплик, и выбирается самое большое, чтобы не было слишком частого уменьшения.
 - Нет ограничений на количество убираемых подов.
2. Масштабирование только одного контроллера. Нельзя настроить несколько HPA, нацеленных на один и тот же Deployment (или StatefulSet), иначе они будут конфликтовать.

Как включить или отключить HPA

HPA не требует отдельного включения в DKP. Но, если необходимо масштабирование не только по метрикам по потреблению CPU и памяти подов, то необходимо включить модуль `prometheus-metrics-adapter`. Как включить модуль см. в документации.

Настройка HPA

Для настройки HPA необходимо:

1. Определить контроллер (Deployment или StatefulSet), который будет масштабироваться.
2. Указать пределы масштабирования (`minReplicas` и `maxReplicas`):

```
minReplicas: 1
maxReplicas: 10
```

3. Сконфигурировать метрики:

```
metrics:
- type: Resource
  resource:
    name: cpu
  target:
    type: Utilization
    averageUtilization: 70
```

4. При необходимости, в параметре `stabilizationWindowSeconds` контроллера настроить время, в течение которого масштабирование может быть отменено, чтобы ограничить скорость прироста подов:

Пример масштабирования вверх:

```
behavior:
  scaleUp:
    stabilizationWindowSeconds: 300
```

Пример масштабирования вниз:

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
```

Примеры настройки НРА

Масштабирование по CPU и памяти

Масштабирование происходит, если среднее использование CPU/памяти всех подов превышает указанный процент:

```

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: app-prod
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  # Границы масштабирования контроллера.
  minReplicas: 1
  maxReplicas: 10
  # Если для приложения характерны кратковременные скачки потребления CPU,
  # можно отложить принятие решения о масштабировании, чтобы убедиться, что оно
необходимо.
  # По умолчанию масштабирование вверх происходит немедленно.
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 300
  metrics:
  # Масштабирование по CPU и памяти.
  - type: Resource
    resource:
      name: cpu
      target:
        # Масштабирование, когда среднее использование CPU всех подов в scaleTargetRef
превышает заданное значение.
        # Для метрики с type: Resource доступен только type: Utilization.
        type: Utilization
        # Масштабирование, если для всех подов из Deployment запрошено по 1 ядру и в
среднем уже используется более 700m.
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        # Пример масштабирования, когда среднее использование памяти всех подов в
scaleTargetRef превышает заданное значение.
        type: Utilization
        # Масштабирование, если для подов запрошено по 1 ГБ памяти и в среднем
использовано уже более 800 МБ.
        averageUtilization: 80

```

Масштабирование по метрикам объектов

Если метрика `rmq-queue-forum-messages` (количество сообщений в RabbitMQ) превышает 42, HPA увеличивает число реплик Deployment `myconsumer`:

```

apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
  (rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"})
  by (<<.GroupBy>>)
  ---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42

```

Масштабирование на основе внешних данных

Используется метрика `mymetric`, получаемая из внешнего источника (например, SQS). Масштабирование происходит, если значение превышает 100:

```

kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  metrics:
    # Используем внешние метрики для масштабирования.
    - type: External
      external:
        metric:
          # Метрика, которую мы зарегистрировали с помощью создания метрики в Prometheus
          kube_adapter_metric_mymetric, но без префикса 'kube_adapter_metric_'.
          name: mymetric
          selector:
            # Для внешних метрик можно и нужно уточнять запрос с помощью лейблов.
            matchLabels:
              namespace: mynamespace
              ingress: myingress
        target:
          # Для метрик типа External можно использовать только `type: Value`.
          type: Value
          # Масштабирование, если значение нашей метрики больше 10.
          value: 10

```

Дополнительные ресурсы

- Документация модуля `observability`
- Документация модуля `prometheus-metrics-adapter`

Вертикальное масштабирование

Как работает вертикальное масштабирование (VPA)

Vertical Pod Autoscaler (VPA) позволяет автоматизировать управление ресурсами контейнеров и значительно повысить эффективность работы приложений. VPA полезен в ситуациях, когда заранее неизвестно, сколько ресурсов потребуется приложению. При использовании VPA и включении соответствующего режима работы, запрашиваемые ресурсы устанавливаются на основе фактического потребления ресурсов, полученных от системы мониторинга. Также возможно настроить систему таким образом, чтобы она только рекомендовала ресурсы, не внося изменений автоматически.

Если нагрузка на приложение меняется в зависимости от времени суток, запросов пользователей или других факторов, VPA автоматически подстраивает выделенные ресурсы. Это позволяет избежать падений из-за нехватки ресурсов или чрезмерного расходования CPU и памяти.

Подробнее о режимах работы VPA, его компонентах и ограничениях можно почитать в разделе «Архитектура».

Как включить или отключить VPA

Включить VPA можно следующими способами:

1. Через ресурс `ModuleConfig` (например, `ModuleConfig/vertical-pod-autoscaler`). Установите параметр `spec.enabled` в значение `true` или `false`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  enabled: true
```

2. Через команду `d8` (в поде `d8-system/deckhouse`):

```
d8 system module enable vertical-pod-autoscaler
```

3. Через веб-интерфейс Deckhouse:

- Перейдите в раздел «Deckhouse - «Модули»;
- Найдите модуль `vertical-pod-autoscaler` и нажмите на него;
- Включите тумблер «Модуль включен».

У модуля `vertical-pod-autoscaler` нет обязательных настроек, то есть можно включить его и не настраивать дополнительно. При этом он будет работать со значениями по умолчанию.

После создания ресурса `VerticalPodAutoscaler` посмотреть рекомендации VPA можно с помощью команды:

```
d8 k describe vpa my-app-vpa
```

В секции `status` будут такие параметры:

- `Target` — рекомендуемое количество ресурсов для пода.
- `Lower Bound` — минимальное рекомендуемое количество ресурсов для приложения.
- `Upper Bound` — максимальное рекомендуемое количество ресурсов для приложения.
- `Uncapped Target` — значение, основанное на крайних метриках без учета истории.

Настройка VPA

1. Создайте конфигурации модуля VPA.

Для настройки VPA нужно создать файл конфигурации для модуля. Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system
```

2. Примените файл конфигурации для VPA с помощью `d8 k apply -f <имя файла конфигурации>`.

Подробнее про настройку лимитов VPA см. в разделе Использование.

Пример настройки модуля

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system

```

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `vertical-pod-autoscaler`

Перезапуск подов при изменении конфигурации

Deckhouse Kubernetes Platform может автоматически перезапускать поды при изменении определенных ресурсов ConfigMap и Secret. Эта возможность реализована на базе проекта Reloader и управляется через аннотации, добавляемые к контроллерам подов (Deployment, DaemonSet, StatefulSet).



Reloader не предназначен для работы в отказоустойчивом режиме.

Далее описаны основные аннотации, позволяющие контролировать перезапуск подов.

Поддерживаемые аннотации

Аннотация	Применяется к объектам	Назначение	Примеры значений
<code>pod-reloader.deckhouse.io/auto</code>	Deployment, DaemonSet, StatefulSet	Автоматический перезапуск подов при изменении всех связанных с ним ConfigMap и Secret (используемых как volume или переменные окружения)	"true", "false"
<code>pod-reloader.deckhouse.io/search</code>	Deployment, DaemonSet, StatefulSet	Перезапуск только при изменении ресурсов с аннотацией <code>pod-reloader.deckhouse.io/match: "true"</code>	"true", "false"
<code>pod-reloader.deckhouse.io/configmap-reload</code>	Deployment, DaemonSet, StatefulSet	Указание конкретных ConfigMap, при изменении которых должен выполняться перезапуск	"some-cm", "some-cm1, some-cm2"
<code>pod-reloader.deckhouse.io/secret-reload</code>	Deployment, DaemonSet, StatefulSet	Указание конкретных Secret, при изменении которых должен выполняться перезапуск	"some-secret", "some-secret1, some-secret2"

Аннотация	Применяется к объектам	Назначение	Примеры значений
<code>pod-reloader.deckhouse.io/match</code>	ConfigMap, Secret	Помечает ресурсы, изменения которых отслеживаются при использовании аннотации <code>pod-reloader.deckhouse.io/search: "true"</code>	"true", "false"



Аннотация `pod-reloader.deckhouse.io/search` не должна использоваться совместно с `pod-reloader.deckhouse.io/auto: "true"`. В этом случае аннотации `pod-reloader.deckhouse.io/search` и `pod-reloader.deckhouse.io/match` будут проигнорированы. Для корректной работы установите `pod-reloader.deckhouse.io/auto: "false"` или удалите её.

Аннотации `pod-reloader.deckhouse.io/configmap-reload` и `pod-reloader.deckhouse.io/secret-reload` не работают при наличии `pod-reloader.deckhouse.io/auto: "true"`. Для корректной работы отключите `auto`.

Примеры использования

Слежение за всеми изменениями во всех подключенных ресурсах

Подключённые ресурсы могут быть использованы как в переменных окружения, так и смонтированы как тома (volumes).

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
  annotations:
    pod-reloader.deckhouse.io/auto: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages

```

Слежение за изменениями только в конкретных ресурсах

Указывает, что нужно отслеживать только те секреты или ConfigMap'ы, которые имеют аннотацию `pod-reloader.deckhouse.io/match: "true"`. Это удобно, если в поде используется множество ресурсов, но перезапуск требуется только при изменении определённых.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/search: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
  annotations:
    pod-reloader.deckhouse.io/match: "true"

```

Слежение за изменениями в ресурсах из списка

Явное указание списка ConfigMap, при изменении которых должен перезапускаться под.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/configmap-reload: "nginx-config,nginx-pages"
spec:
  template:
    spec:
      containers:
        - name: nginx
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
            - name: config
              mountPath: "/etc/nginx/templates"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
        - name: config
          configMap:
            name: nginx-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config

```

Как включить или отключить перезапуск подов

Включить или отключить перезапуск подов можно следующими способами:

1. Через ресурс ModuleConfig (например, ModuleConfig/pod-reloader). Установите параметр `spec.enabled` в значение `true` или `false`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: pod-reloader
spec:
  enabled: true

```

2. Через команду `d8` (в поде `d8-system/deckhouse`):

```
d8 system module enable pod-reloader
```

3. Через веб-интерфейс Deckhouse:

- Перейдите в раздел «Deckhouse - «Модули»;
- Найдите модуль `pod-reloader` и нажмите на него;
- Включите тумблер «Модуль включен».

Настройка

Механизм перезапуска подов работает «из коробки» и не требует обязательной конфигурации. По умолчанию он включён в наборах модулей Default и Managed и отключён в наборе Minimal.

При необходимости его поведение можно изменить в настройках модуля `pod-reloader` (`ModuleConfig pod-reloader`).

Доступные параметры:

Параметр	Тип	Описание	Значение по умолчанию
<code>reloadOnCreate</code>	boolean	Перезапуск при создании ConfigMap или Secret, а не только при изменении	<code>true</code>
<code>nodeSelector</code>	object	Ограничение на узлы для запуска компонента (аналог <code>spec.nodeSelector</code>)	Не задан
<code>tolerations</code>	array	Допуски к размещению на узлах с <code>taint</code> (аналог <code>spec.tolerations</code>)	Не заданы

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `pod-reloader`

Планировщик

Распределение подов

Общее устройство планировщика

Планировщик (scheduler) — это компонент, отвечающий за распределение подов между узлами кластера на основе доступных ресурсов и заданных правил. Он выбирает наиболее подходящий узел для запуска каждого пода, учитывая множество факторов, таких как доступность CPU и памяти, расположение данных, топологию сети, лейблы узлов и специфические требования приложений.

Планировщик работает поэтапно, проходя через несколько фаз, в каждой из которых задействуются различные плагины. Эти плагины оценивают узлы на соответствие заданным критериям и помогают выбрать оптимальный вариант для размещения пода.

Основные задачи планировщика:

- Фильтрация (Filtering) — определение узлов, которые соответствуют требованиям пода.
- Оценка (Scoring) — присвоение баллов узлам на основе различных критериев (чем выше балл, тем лучше узел).
- Принятие решения (Binding) — назначение пода на лучший узел по итогам оценки.

Используемые плагины могут работать в одной или нескольких фазах. Например, один плагин может участвовать только в фильтрации, а другой — как в фильтрации, так и в оценке.

Примеры плагинов:

- `ImageLocality` (фаза: Scoring) — отдает предпочтение узлам, на которых уже есть образы контейнеров, необходимые для пода. Это помогает сократить время развертывания, так как контейнеру не нужно загружать образ с удаленного репозитория.
- `TaintToleration` (фазы: Filtering, Scoring) — реализует механизм `taints` и `tolerations`. Позволяет избежать назначения подов на узлы с нежелательными характеристиками или, наоборот, принудительно размещать их на специальных узлах (например, с выделенными ресурсами).
- `NodePorts` (фаза: Filtering) — проверяет, есть ли у узла свободные порты, необходимые для работы пода. Это особенно важно для сервисов с фиксированными `NodePort`, так как на одном узле не может быть двух подов, использующих один и тот же порт.

С полным списком плагинов можно ознакомиться в документации Kubernetes.

Этапы планирования подов

Процесс назначения подов на узлы в DKP проходит через несколько ключевых этапов. Планировщик анализирует доступные узлы, применяет различные критерии фильтрации и оценки, а затем выбирает наиболее подходящий узел для запуска пода.

На первой фазе Filtering — активируются плагины фильтрации (filter-плагины), которые из всех доступных узлов выбирают те, которые удовлетворяют определенным условиям фильтрации (например, taints, nodePorts, nodeName, unschedulable и другие).

Если узлы распределены по различным зонам (availability zones), DKP чередует выбор зон, чтобы избежать размещения всех подов в одной зоне. Например, если узлы расположены так:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

То выбор будет происходить в таком порядке:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4
```

Важно отметить, что планировщик не проверяет все узлы в кластере, а лишь их часть, что оптимизирует процесс планирования. Количество проверяемых узлов зависит от размера кластера:

- В небольших кластерах (≤ 50 узлов) проверяются все узлы;
- В средних кластерах (100 узлов) — 50% узлов;
- В очень крупных кластерах (5000 узлов) — 10% узлов;
- Минимальный порог — 5% узлов, если их количество превышает 5000.

После завершения этапа фильтрации начинается фаза оценки (Scoring). На этом этапе каждый узел из отфильтрованного списка получает баллы от различных scoring-плагинов, которые анализируют его характеристики и соответствие требованиям пода.

Каждый плагин оценивает узел по своему критерию и присваивает ему определенное количество баллов. Затем все оценки суммируются, формируя общий рейтинг узла.

Основные факторы, которые учитываются при оценке:

- Загрузка узла (pod capacity) – учитывается доступное количество CPU и памяти. Узлы с большим количеством свободных ресурсов получают более высокий балл.
- Локальность образов контейнеров (ImageLocality) – узлы, на которых уже загружены необходимые образы контейнеров, получают преимущество.
- Учет affinity и anti-affinity – если под должен размещаться рядом с определенными подами или, наоборот, быть от них изолирован, узлы, соответствующие этим требованиям, получают более высокий приоритет.
- Доступность хранилища (volume provisioning) – узлы, на которых доступны необходимые PersistentVolumes, оцениваются выше.

После суммирования всех оценок узел с наибольшим итоговым баллом выбирается для запуска пода.

Если несколько узлов получили одинаковую максимальную оценку, выбор происходит случайным образом, чтобы избежать неэффективного предсказуемого распределения нагрузки.

Как изменить или расширить логику работы планировщика

Планировщик можно гибко настраивать и расширять с помощью дополнительных плагинов. Это позволяет адаптировать его поведение под специфические требования кластера, например, учитывать нестандартные метрики, особые правила распределения нагрузки или приоритетность узлов.

Каждый такой плагин представляет собой вебхук, который должен соответствовать следующим требованиям:

- Использование TLS – соединение с плагином должно быть защищено с использованием TLS.
- Доступность – плагин должен быть развернут как сервис внутри кластера и доступен через HTTP(S).
- Поддержка стандартных операций Verbs, filterVerb = filter – участвует в фильтрации узлов перед выбором места для пода, prioritizeVerb = prioritize – участвует в оценке узлов на этапе Scoring.
- Кеширование информации об узлах – предполагается, что все подключаемые плагины могут кэшировать информацию об узле (параметр nodeCacheCapable: true) для повышения производительности.

Подключение расширенного (extender) плагина осуществляется через ресурс

KubeSchedulerWebhookConfiguration – ресурс определяет конфигурацию внешнего вебхука планировщика (kube-scheduler) и позволяет учитывать более сложные условия при планировании нагрузки в кластере, например:

- Размещение подов приложений организации хранилища данных ближе к самим данным,
- Приоритизация узлов в зависимости от их состояния (сетевой нагрузки, состояния подсистемы хранения и т.д.),
- Разделение узлов на зоны, и т.п.

Пример подключения внешнего плагина планировщика через вебхук:

```
apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
  timeoutSeconds: 5
```

✗ Если используется параметр `failurePolicy: Fail`, то при сбое работы вебхука планировщик полностью прекратит свою работу, и новые поды не смогут быть запущены. Рекомендуется тщательно тестировать работу расширяемых плагинов перед развертыванием в продуктивной среде.

Дополнительные механизмы управления размещением подов

Deckhouse Kubernetes Platform предоставляет гибкие механизмы для управления размещением подов в кластере. Эти механизмы позволяют оптимизировать балансировку нагрузки, повышать отказоустойчивость и изолировать системные и пользовательские нагрузки.

Основные способы управления размещением подов:

- Использование лейблов узлов (`NodeGroup.spec.nodeTemplate.labels`). Позволяет явно указать, на каких узлах должны запускаться определенные поды, с помощью `spec.nodeSelector` или `spec.affinity.nodeAffinity`.
 - Настройка `taints` и `tolerations`. Позволяет ограничить запуск подов только на определенных узлах, предотвращая их размещение на неподходящих хостах.
 - Использование пользовательских `toleration`-ключей (`settings.modules.placement.customTolerationKeys`). Позволяет управлять запуском критически важных компонентов Deckhouse (например, CNI и CSI) на выделенных узлах.
1. Использование `nodeSelector` — позволяет явно указывать узлы, на которых должны запускаться поды. Это делается путем назначения лейблов узлам и последующего использования `spec.nodeSelector` в манифесте пода. Пример:

Допустим, у нас есть узел `kube-system-1`, предназначенный для сервисов мониторинга. Добавим на него лейбл:

```
d8 k label node kube-system-1 node-role/monitoring=""
```

Теперь, чтобы развернуть поды **только** на этом узле, в Deployment добавляется `nodeSelector`:

```
nodeSelector:
  node-role/monitoring: ""
```

2. Использование `taints` и `tolerations` — в отличие от `nodeSelector`, `taints` позволяют запретить запуск подов на узлах, если у них нет соответствующего `toleration`. Это гарантирует, что на

узле будут запускаться только определенные сервисы. Пример:

Например, есть узел `kube-frontend-1`, который предназначен только для Ingress-контроллеров. Добавим на него taint:

```
d8 k taint node kube-frontend-1 node-role/frontend="":NoExecute
```

Теперь на этом узле не будет запускаться ничего, кроме подов, у которых есть соответствующий toleration:

```
tolerations:
- effect: NoExecute
  key: node-role/frontend
```

Это позволяет защитить узел от случайного запуска на нем других подов, не относящихся к его предназначению.

- DKP поддерживает механизм `customTolerationKeys`, который позволяет явно определять допустимые toleration-ключи. Это полезно, если требуется разместить системные сервисы (CNI, CSI и другие) на определенных узлах. Пример:

```
customTolerationKeys:
- dedicated.example.com
- node-dedicated.example.com/master
```

Этот механизм позволяет:

- Разделять узлы на зоны, выделяя одни под Ingress-контроллеры, другие под системные сервисы (Prometheus, VPN, CoreDNS).
- Изолировать критически важные приложения от системных компонентов, избегая конкуренции за ресурсы.

Профили планировщика

Планировщик поддерживает несколько профилей, которые определяют стратегию размещения подов в кластере. В зависимости от выбранного профиля поды будут распределяться на узлы с разной логикой балансировки нагрузки.

- `default-scheduler` — стандартный профиль, который старается равномерно распределять поды по узлам, отдавая предпочтение менее загруженным.
- `high-node-utilization` — профиль, который размещает поды на более загруженных узлах. Это может быть полезно в сценариях, когда необходимо уплотнить нагрузку и освободить неиспользуемые узлы для дальнейшего выключения или перераспределения ресурсов.

Для выбора профиля планировщика укажите его в параметре `spec.schedulerName` манифеста пода. Пример:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
  - name: example-pod
    image: registry.k8s.io/pause:2.0
```

Перераспределение подов

DKP каждые 15 минут анализирует состояние кластера и выполняется вытеснение (eviction) подов, которые соответствуют условиям, описанным в активных стратегиях планирования. Вытесненные поды вновь проходят стандартный процесс планирования с учётом текущего состояния кластера. Это позволяет перераспределить рабочие нагрузки в соответствии с выбранной стратегией и, при необходимости, освободить ресурсы некоторых узлов.

Учет класса приоритета пода

DKP создает набор классов приоритетов, которые определяют важность подов в кластере и порядок их вытеснения при перераспределении нагрузки.

Если в кластере не хватает ресурсов, поды с низким приоритетом могут быть вытеснены в пользу более важных подов. Это помогает гарантировать работу критически важных сервисов, даже если узлы перегружены.

Как DKP учитывает класс приоритета

Deckhouse Kubernetes Platform использует механизм приоритетов подов для определения порядка их вытеснения при нехватке ресурсов. Чем выше приоритет пода, тем меньшая вероятность его удаления в процессе перераспределения нагрузки.

Для управления этим механизмом используется модуль `descheduler`, в манифесте которого можно задать порог приоритета с помощью параметра `spec.priorityClassThreshold`. Это позволяет ограничить вытеснение подов, имеющих приоритет ниже заданного значения.

Порог можно указать двумя способами:

- По названию класса: если указать `priorityClassThreshold.name`, то вытеснению будут подлежать только поды с приоритетом ниже указанного класса приоритета.
- По числовому значению: если указать `priorityClassThreshold.value`, то вытеснению подлежат поды с приоритетом ниже заданного значения (целочисленное значение).

Пример:

```
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: custom
spec:
  priorityClassThreshold:
    name: high-priority
```

Какие поды не вытесняются

- DKP не вытесняет под в следующих случаях:
 - Под находится в пространстве имен `d8-*` или `kube-system`;
 - Под имеет `priorityClassName` `system-cluster-critical` или `system-node-critical`;
 - Под связан с локальным хранилищем;
 - Под связан с `DaemonSet`;
 - Вытеснение пода нарушит Pod Disruption Budget (PDB);
 - Нет доступных узлов для запуска вытесненного пода.

Если несколько подов подходят под критерии вытеснения, DKP применяет дополнительную логику:

1. Сначала вытесняются поды с наименьшим приоритетом (`BestEffort`);
2. Затем — поды с приоритетом `Burstable`;
3. В последнюю очередь вытесняются `Guaranteed` поды, если это возможно.

DKP позволяет точно настроить, на какие поды и узлы будет распространяться вытеснение:

- `spec.podLabelSelector` — ограничивает поды по лейблам;
- `spec.namespaceLabelSelector` — фильтрует пространства имен, из которых поды будут рассматриваться для вытеснения;
- `spec.nodeLabelSelector` — выбирает узлы по лейблам.

Каждый из этих полей содержит стандартные поля `matchExpressions` и `matchLabels`, в которых можно указывать операторы `In`, `NotIn`, `Exists`, `DoesNotExist` и нужные значения лейблов.

Как включить или отключить перераспределение

Для включения функции перераспределения подов необходимо включить модуль `descheduler`.

Это можно сделать следующими способами:

1. Через ресурс `ModuleConfig` (например, `ModuleConfig/descheduler`). Установите параметр `spec.enabled` в значение `true` или `false`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true    # или false для отключения
```

2. Через команду `d8` (в поде `d8-system/deckhouse`):

```
d8 system module enable descheduler
# или, чтобы отключить:
d8 system module disable descheduler
```

3. Через веб-интерфейс Deckhouse:

- Перейдите в раздел «Deckhouse - «Модули»;
- Найдите модуль `descheduler` и нажмите на него;
- Включите тумблер «Модуль включен».

У модуля нет обязательных настроек, то есть можно включить его и не настраивать дополнительно. При этом он будет работать со значениями по умолчанию.

Стратегии перераспределения

В параметре `spec.strategies` перечисляются стратегии, которые вы хотите включить или настроить. Каждая стратегия имеет флаг `enabled` (по умолчанию `false`).

Ниже приведён список основных стратегий, доступных в DKP.

HighNodeUtilization — концентрирует нагрузку на меньшем числе узлов, вытесняя поды с недостаточно нагруженных узлов, чтобы они перезапустились на других узлах. Требуется:

- Специальная настройка модуля `descheduler` — `MostAllocated`;
- (Опционально) включенный автоскейлинг кластера — чтобы неиспользуемые узлы могли быть выключены.

Стратегия включается параметром `spec.strategies.highNodeUtilization.enabled`.

Параметр `thresholds` задаёт порог для признания узла «недостаточно нагруженным». Если ресурс (CPU, память и т.д.) ниже всех пороговых значений, узел считается недогруженным, и DKP попытается вытеснить с него поды.

Пример:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

Если какой-либо из этих типов ресурсов не указан, все соответствующие пороги по умолчанию устанавливаются в 100, чтобы избежать ситуации, когда узлы переходят из состояния недоиспользованных в перегруженные. Т.е. если параметру `thresholds` не указать порог `memory`, то он по умолчанию установится в 100.

```
thresholds:
  cpu: 50
```



В GKE (Google Kubernetes Engine) нельзя настроить MostAllocated по умолчанию, но можно использовать стратегию `optimize-utilization`.

LowNodeUtilization — более равномерно нагружает узлы. Стратегия выявляет недостаточно нагруженные узлы и вытесняет поды с других, избыточно нагруженных узлов. Стратегия предполагает, что пересоздание вытесненных подов произойдет на недостаточно нагруженных узлах.

Стратегия включается параметром `spec.strategies.lowNodeUtilization.enabled`.

Недостаточно нагруженный узел — узел, использование ресурсов которого меньше всех пороговых значений, заданных в секции параметров `strategies.lowNodeUtilization.thresholds`.

Избыточно нагруженный узел — узел, использование ресурсов которого больше хотя бы одного из пороговых значений, заданных в секции параметров `strategies.lowNodeUtilization.targetThresholds`.

Узлы с использованием ресурсов в диапазоне между `thresholds` и `targetThresholds` считаются оптимально используемыми. Поды на таких узлах вытесняться не будут.

Пример:

```
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

RemoveDuplicates — не позволяет нескольким подам одного контроллера (ReplicaSet/ReplicationController/StatefulSet/Job) одновременно находиться на одном узле (исключая DaemonSet). Если по каким-то причинам на одном узле оказалось 2+ подов от одного контроллера, стратегия вытеснит «лишние» поды, чтобы они равномернее распределились по кластеру.

Ситуация может возникать после кратковременного выхода из строя узлов, когда поды «переехали», а узел вернулся в строй, но поды так и остались сконцентрированы на другом узле.

Стратегия включается параметром `strategies.removeDuplicates.enabled`.

Пример:

```
spec:
  strategies:
    removeDuplicates:
      enabled: true
```

RemovePodsHavingTooManyRestarts — вытесняет с узлов поды со слишком большим количеством перезапусков. Кандидатами на вытеснение становятся поды, у которых суммарное число перезапусков всех контейнеров, включая init-контейнеры, превышает порог `podRestartThreshold`. Полезна для вытеснения подов в состоянии `CrashLoopBackOff` или с постоянными сбоями, а также для освобождения ресурсов и возможности планирования новых подов на потенциально более здоровых узлах.

Стратегия включается параметром `strategies.removePodsHavingTooManyRestarts.enabled`.

Пример:

```

список:
  стратегии:
    removePodsHavingTooManyRestarts:
      enabled: true

```

RemovePodsViolatingInterPodAntiAffinity — вытесняет любые поды, нарушающие правила `inter-pod affinity/anti-affinity`. Например, если под2 и под3 имеют `anti-affinity` по отношению к под1, но все трое по каким-то причинам оказались на одном узле, модуль вытеснит под1, чтобы под2 и под3 могли продолжать работу в соответствии со своими правилами.

Стратегия включается параметром `strategies.removePodsViolatingNodeAffinity.enabled`.

Пример:

```

список:
  стратегии:
    removePodsViolatingInterPodAntiAffinity:
      enabled: true

```

RemovePodsViolatingNodeAffinity — эта стратегия отвечает за вытеснение подов, которые больше не соответствуют своим требованиям `node affinity`. `Node affinity` определяет, какие узлы подходят поду при размещении:

1. `requiredDuringSchedulingIgnoredDuringExecution`:
 - Под с таким типом правила обязан запускаться только на узле, удовлетворяющем заданным условиям (например, наличие определённого лейбла).
 - Если со временем узел перестаёт удовлетворять этим условиям (например, лейбл убрали), а в кластере есть другой узел, который им по-прежнему соответствует, DKP вытеснит под, чтобы он заново запустился на подходящем узле.
2. `preferredDuringSchedulingIgnoredDuringExecution`:
 - Под с таким типом правила может быть размещён на узле, который удовлетворяет предпочтению, но если подходящего узла не было, он запускается там, где мог.
 - Если позже в кластере появляется более соответствующий узел, DKP может вытеснить под со старого узла, чтобы он перезапустился уже оптимальных условиях.

Таким образом, стратегия `RemovePodsViolatingNodeAffinity` помогает поддерживать соответствие подов их актуальным правилам привязки к узлам и гарантировать, что при появлении более подходящих вариантов поды не застрянут в неудачном местоположении.

Стратегия включается параметром `список.стратегии.removePodsViolatingNodeAffinity.enabled`.

Пример:

```

список:
  стратегии:
    removePodsViolatingNodeAffinity:
      enabled: true
      nodeAffinityType:
        - requiredDuringSchedulingIgnoredDuringExecution
        - preferredDuringSchedulingIgnoredDuringExecution

```

RemovePodsViolatingTopologySpreadConstraint — гарантирует, что поды, нарушающие ограничения распределения по топологии, будут вытеснены с узлов. Вытесняется минимальное количество подов, необходимое для приведения доменов топологии в соответствие с `maxSkew` каждого ограничения. Стратегия полезна для перебалансировки подов между зонами доступности после восстановления зоны из аварии.

Стратегия включается параметром `стратегии.removePodsViolatingTopologySpreadConstraint.enabled`.

Пример:

```

spec:
  strategies:
    removePodsViolatingTopologySpreadConstraint:
      enabled: true

```

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `console`
- Кастомные ресурсы модуля `control-plane-manager`
- Документация модуля `descheduler`
- Кастомные ресурсы модуля `node-manager`

Классы приоритета

Deckhouse Kubernetes Platform (DKP) создаёт в кластере набор классов приоритета (PriorityClass) и назначает их своим компонентам. Приложения могут использовать эти классы через параметр `priorityClassName`. Планировщик (scheduler) учитывает приоритет пода: если ресурсов недостаточно, первыми вытесняются поды с более низким приоритетом.

Например, если подам нужен класс `production-low`, а ресурсов не хватает, планировщик сначала вытеснит поды с более низким классом приоритета — например, с `develop`, затем с `cluster-low` и так далее. Если `priorityClassName` не указан, под считается наименее приоритетным.

Доступные классы приоритета

В таблице перечислены классы приоритета, которые DKP создаёт в кластере, в порядке убывания приоритета (чем больше значение, тем выше приоритет). Выбирайте класс с учётом окружения и критичности нагрузки.

✗ Не используйте классы приоритета `system-node-critical`, `system-cluster-critical`, `cluster-medium`, `cluster-low`, так как они зарезервированы для ключевых компонентов кластера.

Класс приоритета	Описание	Значение
<code>system-node-critical</code>	Компоненты кластера, которые обязаны присутствовать на узле. Также полностью защищает от вытеснения kubelet'ом. Примеры: <code>node-exporter</code> , <code>csi</code> и другие.	2000001000
<code>system-cluster-critical</code>	Компоненты кластера, без которых его корректная работа невозможна. Этим классом приоритета обязательно помечаются <code>MutatingWebhooks</code> и <code>Extension API servers</code> . Также полностью защищает от вытеснения kubelet'ом. Примеры: <code>kube-dns</code> , <code>kube-proxy</code> , <code>cni-flannel</code> , <code>cni-cilium</code> и другие.	2000000000
<code>production-high</code>	Stateful-приложения, отсутствие которых в production-окружении приводит к полной недоступности сервиса или потере данных. Примеры: <code>PostgreSQL</code> , <code>Memcached</code> , <code>Redis</code> , <code>MongoDB</code> и другие.	9000

Класс приоритета	Описание	Значение
<code>cluster-medium</code>	Компоненты кластера, влияющие на мониторинг (алерты, диагностика) и автомасштабирование. Без мониторинга невозможно оценить масштабы происшествия, без автомасштабирования — предоставить приложениям необходимые ресурсы. Примеры: <code>deckhouse</code> , <code>node-local-dns</code> , <code>grafana</code> , <code>upmeter</code> и другие.	7000
<code>production-medium</code>	Основные stateless-приложения в production-окружении, которые отвечают за работу сервиса для посетителей.	6000
<code>deployment-machinery</code>	Компоненты кластера, используемые для сборки и развёртывания в кластер.	5000
<code>production-low</code>	Приложения в production-окружении (cron-задания, административные панели, batch-процессы), без которых можно обойтись некоторое время. Если batch или cron-задачи нельзя прерывать, их следует отнести к <code>production-medium</code> .	4000
<code>staging</code>	Staging-окружения для приложений.	3000
<code>cluster-low</code>	Компоненты кластера, без которых эксплуатация возможна, но которые желательны. Примеры: <code>dashboard</code> , <code>cert-manager</code> , <code>prometheus</code> и другие.	2000
<code>develop</code> (по умолчанию)	Develop-окружения для приложений. Класс по умолчанию, если не указан иной класс.	1000
<code>standby</code>	Класс не предназначен для приложений. Используется в системных целях для резервирования узлов.	-1

Создание дополнительных классов приоритета

Помимо классов, которые DKP создаёт в кластере, можно добавить дополнительный класс приоритета: задать имя и числовой приоритет.

Далее приведён пример создания класса приоритета `critical-applications` с приоритетом 8000. При использовании этого класса приоритета соответствующие поды будут вытеснены перед подами с классом приоритета `production-high` (и выше), но после подов с классом приоритета `cluster-medium` (и ниже).

Создайте файл `critical-applications.yaml` со следующим содержимым:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: critical-applications
value: 8000
globalDefault: false
description: "Приоритет для критичных приложений"
```

В манифесте ресурса PriorityClass используются следующие поля:

- `metadata.name` — имя класса. Указывается в подах в параметре `priorityClassName`.
- `value` — числовой приоритет. Чем выше значение, тем выше приоритет.
- `globalDefault` — определяет, будет ли этот класс использоваться по умолчанию для подов без явного `priorityClassName`.
- `description` — описание класса.



Будьте осторожны с параметром `globalDefault: true`. Если установить его для дополнительного класса, все поды в кластере без явного приоритета получат это значение, что может привести к непредсказуемому вытеснению системных компонентов.

Примените манифест в кластере:

```
d8 k apply -f critical-applications.yaml
```

Проверьте создание класса приоритета:

```
d8 k get priorityclass critical-applications
```

Пример вывода:

NAME	VALUE	GLOBAL-DEFAULT	AGE	PREEMPTIONPOLICY
critical-applications	8000	false	7s	PreemptLowerPriority



Не создавайте дополнительные классы со значениями выше 1 000 000, чтобы не нарушить работу критически важных системных компонентов.

Механизм вытеснения

Принцип работы

Решение о вытеснении принимается планировщиком исключительно на основе числового значения приоритета. Тип рабочей нагрузки (Deployment, StatefulSet, DaemonSet или обычный под) не имеет значения — планировщик сравнивает только числа: под с большим значением приоритета может вытеснить под с меньшим, чтобы освободить ресурсы на узле.

При равных приоритетах вытеснение не происходит. Если новый под имеет такой же приоритет, как и существующие поды на узле, планировщик не будет вытеснять их — новый под останется в статусе `Pending` до появления свободных ресурсов.

Практические сценарии и пошаговые демонстрации смотрите в разделе «Использование классов приоритета».

Защита пода от вытеснения

Единственный способ защитить под от вытеснения — назначить ему достаточно высокий приоритет. Однако даже при высоком приоритете под будет вытеснен, если в кластере появится под с ещё более высоким приоритетом.

Для снижения рисков при вытеснении используется комбинация трёх механизмов. Эти механизмы универсальны и применимы к любым критичным приложениям:

1. Указание класса с высоким приоритетом в параметре `priorityClassName` — делает под менее вероятным кандидатом на вытеснение.

2. Использование PodDisruptionBudget (PDB) — ограничивает количество одновременно удаляемых реплик. При вытеснении PDB работает как рекомендация (в отличие от плановых работ, таких как drain узла, когда PDB является жёстким ограничением).
3. `terminationGracePeriodSeconds` — даёт приложению время на сброс буферов и закрытие соединений перед принудительным удалением. Это последняя линия обороны, гарантирующая целостность данных даже при нарушении PDB.



Без параметра `terminationGracePeriodSeconds` у пода мгновенное завершение работы может привести к потере данных, которые не были сохранены на диск, и, например, к повреждению базы данных.

Отличия классов приоритета от других механизмов управления ресурсами

Классы приоритета (PriorityClass) часто путают с другими механизмами управления ресурсами. Важно понимать их различия:

Механизм	Назначение	Отличие от классов приоритета
ResourceQuota	Лимиты на потребление ресурсов в неймспейсе	Ограничивает суммарное потребление, не влияет на вытеснение подов.
LimitRange	Лимиты и запросы для подов по умолчанию	Задаёт минимальные и максимальные значения, не влияет на вытеснение.

В отличие от этих механизмов, классы приоритета влияют именно на порядок вытеснения подов при нехватке ресурсов, а не на ограничение потребления.

Эксплуатация и диагностика

Если под с нужным классом приоритета не запускается, хотя в кластере в целом есть свободные ресурсы, проблема часто связана не с самим классом, а с размещением нагрузки.

Возможные причины на уровне кластера:

- Свободные CPU и память есть, но они распределены по разным узлам, и ни один узел не может принять под целиком даже после вытеснения.
- Нет подходящих кандидатов на вытеснение — на целевом узле все поды имеют равный или более высокий приоритет. Как проверить — в пользовательском разделе «Практическая проверка отсутствия подходящих подов».
- Нет подходящих узлов для размещения пода — например, из-за использования taints и tolerations.
- Достигнут лимит подов на узле (`Capacity.pods`) — вытеснение не помогает: число подов на узле не уменьшается, вытесненный под заменяется новым.

Что можно сделать:

1. Добавьте worker-узлы или перераспределите нагрузку, если свободные ресурсы распределены по разным узлам.
2. Освободите место на перегруженном узле: удалите или перенесите менее критичные поды.
3. Пересмотрите плотность DaemonSet и системных подов на узле.
4. При необходимости назначьте приложению более высокий класс приоритета из раздела «Доступные классы приоритета».

Дополнительные ресурсы

- Использование классов приоритета

Сеть

В отличие от базовой установки Kubernetes, в Deckhouse Kubernetes Platform (DKP) настройка сети осуществляется через модули и кастомные ресурсы, что упрощает и стандартизирует конфигурацию.

Платформа поддерживает:

- управление входящим и исходящим трафиком,

- настройка межкластерного взаимодействия,
- управление сетевыми политиками и маршрутизацией,
- управление DNS в кластере Kubernetes.

Настройка и управление входящим трафиком

Управление входящим трафиком осуществляется с помощью Ingress-контроллеров. Они маршрутизируют запросы пользователей к нужным приложениям и сервисам на основе правил, описанных в ресурсах Ingress. В DKP возможно управление трафиком как на сетевом уровне (NLB – Network Load Balancer), так и на прикладном (ALB – Application Load Balancer).

Управление исходящим трафиком

Администратор DKP имеют возможность настраивать исходящий трафик и управлять им, чтобы гарантировать корректную обработку и маршрутизацию всех исходящих данных.

Настройка исходящего трафика позволяет контролировать и фильтровать соединения, что особенно важно для обеспечения безопасности при интеграции с внешними API и сервисами. Для этого можно использовать политики, определяющие правила разрешения или блокировки трафика.

Управление исходящим трафиком обеспечивается за счет использования Egress Gateway (модуль `cni-cilium`), а также инструментов Istio (модуль `istio`).

Настройка внутренней сети

DKP предлагает широкий спектр возможностей для управления внутренней сетью. Администратор может настраивать взаимодействие между подами и узлами кластера и шифрование трафика с помощью различных технологий и инструментов.

Настройка межкластерного взаимодействия

DKP поддерживает два подхода к организации взаимодействия между суверенными и созависимыми кластерами:

- Мультикластер — объединение кластеров для совместного использования ресурсов, балансировки нагрузки и повышения отказоустойчивости. Подходит для распределённых команд и инфраструктур в разных регионах.
- Федерация — централизованное управление несколькими суверенными кластерами с едиными политиками и синхронизацией конфигураций. Эффективна для крупных организаций и комплексных систем.

Подробнее о настройке межкластерного взаимодействия — в разделе Межкластерное взаимодействие.

Сетевые политики

С помощью сетевых политик в DKP определяются правила, регулирующие поток трафика между подами, узлами, пространствами имен и внешними системами. Сетевые политики обеспечивают изоляцию подов, защиту от атак внутри кластера, контролировать доступ к внешним сервисам, входящие и исходящие соединения.

Дополнительные ресурсы

- Параметры модуля `cni-cilium`
- Документация модуля `istio`

Настройка внутренней сети

В Deckhouse Kubernetes Platform (DKP) для настройки сети используются CNI-плагины. Рекомендуемый вариант — Cilium, подходит для большинства сценариев. Также поддерживаются CNI Flannel и Simple Bridge (обычно применяются для облачных провайдеров).

Сетевые параметры задаются при развёртывании кластера DKP:

- указываются адресные пространства для подов и сервисов кластера;
- выбирается режим работы сети для Cilium (можно изменить позже).

Режимы работы внутренней сети

Режим работы определяется параметром `tunnelMode` в настройках модуля `cni-cilium`.

Поддерживается два режима:

- классический,
- с использованием VXLAN-туннеля.

Выбор подходящего варианта зависит от особенностей сети кластера:

- Если узлы кластера находятся в одной сети, могут быть использованы оба варианта.
- Если узлы кластера находятся в разных сетях, рекомендуется использовать VXLAN для создания виртуальной изолированной сети поверх существующей. При этом обеспечивается возможность работы с различными сетевыми сегментами.

Различия в производительности между режимами минимальны. Особенности режимов приведены в таблице.

Параметр	Классический режим	Режим с VXLAN-туннелем
Тип сети	Для узлов в одной сети	Подходит для узлов в разных сетях
Изоляция трафика	Нет	Обеспечивается
Использование инфраструктуры	Прямая маршрутизация	Создание виртуальной сети поверх существующей
Изменение MTU	Нет изменений	Уменьшение MTU из-за инкапсуляции
Дополнительная маршрутизация	Добавляется промежуточный узел	Нет необходимости

Настройка режима

Режим работы внутренней сети настраивается через ресурс ModuleConfig с именем `cni-cilium`.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings:
    tunnelMode: VXLAN
```

Подробнее о настройках модуля `cni-cilium` – в документации.



После смены режима работы перезагрузите все узлы, иначе возможны проблемы с доступностью подов.

Сервисы

Сервисы в DKP реализуются средствами выбранного CNI, либо через `kube-proxy`.

Они обеспечивают доступ к группам подов, выполняющих одинаковые функции, и балансируют трафик между ними. Каждый объект Service задаёт:

- логическую группу конечных точек (подов),
- способ маршрутизации к ним.

Поддерживаемые типы сервисов

- **ClusterIP** — создаёт внутренний IP-адрес, который выбирается из сети сервисов. Весь трафик с этого IP перенаправляется на поды, указанные в селекторе. Тем самым обеспечивается балансировка.

В этом типе сервиса можно отказаться от назначения IP-адреса. Тогда адрес из сети

сервисов не будет назначаться, а сам сервис при обращении будет резолвиться в IP-адреса подов. Балансировка будет работать на уровне DNS (round-robin).

- **NodePort** — открывает указанный порт на каждом узле кластера и перенаправляет входящий трафик на поды. Из соображений безопасности в DKP порт слушается только на внутреннем IP узлов. Это поведение можно изменить, если добавить аннотацию `node.deckhouse.io/nodeport-bind-internal-ip: "false"` на группу узлов.
- **LoadBalancer** — создаётся в облачном провайдере, на котором установлена DKP, и принимает внешний трафик. В bare-metal-кластерах аналогичный функционал реализуется с помощью модуля `metalLB`.
- **ExternalName** — DNS-запись для обращения к сервису (по сути, CNAME).

Настройка сети сервисов через kube-proxy

Если используется Flannel или Simple Bridge, для работы сервисов требуется модуль `kube-proxy`. Он заменяет стандартные компоненты kube-adm (DaemonSet, ConfigMap, RBAC) своей конфигурацией.

По умолчанию в целях безопасности при использовании сервисов с типом NodePort подключения принимаются только на InternalIP узлов. Это поведение можно изменить через аннотацию на узле — `node.deckhouse.io/nodeport-bind-internal-ip: "false"`.

Пример аннотации для NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
```



После добавления, удаления или изменения значения аннотации необходимо вручную перезапустить поды kube-proxy.

При включении модуля `cni-cilium` модуль `kube-proxy` отключается автоматически.

Дополнительные ресурсы

- Документация модуля `cni-cilium`
- Параметры модуля `metalLB`

Шифрование трафика между подами

Для шифрования трафика между подами в Deckhouse Kubernetes Platform (DKP) можно использовать mTLS, реализованный средствами Istio (модуль `istio`).

mTLS (mutual TLS) обеспечивает взаимную аутентификацию сервисов с использованием TLS-сертификатов: при всех исходящих запросах проверяется серверный сертификат, а при входящих — клиентский. После успешной проверки sidecar-proxy идентифицирует удаленный узел и может использовать эти данные для авторизации либо в прикладных целях.

Каждый сервис получает собственный идентификатор в формате `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`.

Здесь:

- `TrustDomain` — домен кластера.
- `ServiceAccount` — имя аккаунта (может быть `default` или заданное пользователем).

Этот идентификатор используется в качестве удостоверяемого имени в TLS-сертификатах.

Настройки можно переопределить на уровне неймспейсов (namespace).

Пример настройки mTLS

1. Включите модуль `istio`:

```
d8 k create -f -<<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 2
  enabled: true
EOF
```

2. Создайте неймспейс, добавьте лейблы:

```
d8 k create namespace test-istio-mtls
d8 k label namespace test-istio-mtls istio-injection=enabled
d8 k label namespace test-istio-mtls security.deckhouse.io/pod-policy=privileged
```

3. Добавьте политику, определяющую режим работы mTLS. Возможные режимы:

- **PERMISSIVE** (используется по умолчанию) — зашифрованный трафик между подами разрешён, но допускается и незашифрованный.
- **STRICT** — допускается только зашифрованный трафик.

Пример применения режима **PERMISSIVE**:

```
d8 k -n test-istio-mtls create -f -<<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
spec:
  mtls:
    mode: PERMISSIVE
EOF
```

4. Создайте Deployment:

- Если используется публичный registry:

```
d8 k -n test-istio-mtls create deployment webserver --image=docker.io/library/nginx:1.26-alpine --port 80
```

- Если используется all-in-one image (замените адрес на свой):

```
d8 k -n test-istio-mtls create deployment webserver --
image=registry.company.network/localrepo/all-in-one-image:0.1 --port 80 -- /bin/sh -c
'nginx -g "daemon off;"'
```

5. Опубликуйте Deployment:

```
d8 k -n test-istio-mtls expose deployment webserver --port=80
```

6. Получите IP-адрес пода и имя узла, на котором он работает. Эта информация понадобится для запуска **tcpdump** на соответствующем узле:

```
d8 k -n test-istio-mtls get pods -l app=webserver -o wide
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE		NOMINATED	NODE	READINESS	GATES
webserver-76d6c9b8c-9mdtb	2/2	Running	0	48m	10.111.1.122
test-worker-e36e4712-5948b-sp9t8		<none>		<none>	

7. Подключитесь к узлу по SSH от root-пользователя и запустите `tcpdump`:

```
tcpdump -A -v -i any host 10.111.1.122 and port 80
```

8. Создайте Deployment client в этом же неймспейсе, чтобы он был участником mesh-сети:

- При использовании образов из публичных registry:

```
d8 k -n test-istio-mtls create deployment client --image=docker.io/library/alpine:3.21 -- /bin/sh -c "sleep infinity"
```

- При использовании образа all-in-one-image (замените адрес на свой):

```
d8 k -n test-istio-mtls create deployment client -- image=registry.company.network/localrepo/all-in-one-image:0.1 -- /bin/sh -c "sleep infinity"
```

9. После создания пода сделайте запрос к сервису `webserver`:

```
d8 k -n test-istio-mtls exec -ti deployments/client -- wget -S --spider --timeout 1 webserver`
```

В `tcpdump` будет виден только зашифрованный трафик.

```
10.111.91.68.http > 10.111.91.167.54204: Flags [P.], cksum 0xd7c7 (incorrect -> 0xe83a), seq
1:3033, ack 518, win 134, options [nop,nop,TS val 3160034944 ecr 2239691194], length 3032: HTTP
E....]@.@.T.
o[D
o[...P..'....;i8.....
.ZN..~.....Z...v...T.....d.4p
..6j..V..G...0 d..U h...e...W.`.....4....;.....T.....3.$...
[...%ex.....i<...*.[...C...jC...p.+.....Nt .....
{..P%.+.,L.....A.....B..7}.v../(...rg.o.)5.....E.X;K8.%,..yb.^f...^+..Ble.
j..w3 S.....}..
|3...+.=...;H..#...Y...3h.....:9.w.A.4X..g...|S;.`<....%.y..4.....D.m...../6.7
[.....!+.J.....2..r.D.C>`.3A..... ..H.^1...P1N....).%...
r.=..s.FB..i^..^....4{;...ED_\....g`l,q.....U*0.#..(.....).h..|sc...
.c(..kd....5C@..5.Z.L..C2
@.7>..y%....R.5.....j9.....!.....H.a....b..].0..~..0x7..+N...o.6.....
.. (%7:.Xr..N.....'.0.a.j&v..Ba....t..q.
=U.t.fU.).....2g....Eat.D.3n.....*G.N..!LY.....n).....jL..9RD..gT.lX..p&..=.d..Tq%....qF
`.....'..|..$!g..j.d. N%tb^...@- ..`...S::D.....y...ckA.;}Y,....X..6...=
[,...|AD..._}.....W... ..u3G...<. ..&.0j.,.'!.#
....w..bx.....}U.;..y....J.K..fQ.#]..3.V..=.d.._'....q.;!9.N.....n.7.Zi.>....@.
..].u.A}.;;....c..s.....d.*=..G..9.....Nt@...v..s.>.
...h....Cm.Z.....n.....L.....-
_.....r...%Z.....h.....`..8A....yt.t..2d....oH?.1.0&.J....F..b.OV.....
.....E1H..%~..2.H..{.I...=.I.*..2y1p0h.....P.....@r....vk.!"...
{..`3..<.,r&L....M...t...;z...Q...1+.,.....:..L.V.....e..
..!|\..6*}....vT.A>....A0.....{.?}AH.+R..g.
[=?..X...|94....S+!..e...*.M`f...o..b.K#5.....@...5.....o.(.
...
```

10. Создайте пространство без лейбла `istio-injection=enabled` и выполните запрос к сервису `webserver.test-istio-mtls` (т.е. выполняется запрос из пода, который не является участником mesh-сети):

```
d8 k create namespace test-istio-mtls-without-injection
```

11. Добавьте Deployment:

- При использовании образов из публичных registry:

```
d8 k -n test-istio-mtls-without-injection create deployment alpine --  
image=docker.io/library/alpine:3.21 -- /bin/sh -c "sleep infinity"
```

- При использовании образа all-in-one-image (замените адрес на свой):

```
d8 k -n test-istio-mtls create deployment client --  
image=registry.company.network/localrepo/all-in-one-image:0.1 -- /bin/sh -c "sleep  
infinity"
```

12. После создания пода сделайте запрос к сервису `webserver.test-istio-mtls`:

```
d8 k -n test-istio-mtls-without-injection exec -ti deployments/alpine -- wget -S --spider --  
timeout 1 webserver.test-istio-mtls`.
```

В выводе `tcpdump` будут отображаться незашифрованные (plain text) запросы и ответы:

```

10.111.91.211.54424 > 10.111.91.68.http: Flags [P.], cksum 0xcc77 (incorrect -> 0xee57), seq
1:93, ack 1, win 128, options [nop,nop,TS val 305286128 ecr 3834507600], length 92: HTTP, length:
92
GET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

E...y.@.?...
o[.
o[D...P....<Nb.....w.....
.2K....PGET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

09:14:20.960302 lxc4f4a182c887c In IP (tos 0x0, ttl 64, id 33396, offset 0, flags [DF], proto TCP
(6), length 52)
10.111.91.68.http > 10.111.91.211.54424: Flags [.], cksum 0xcc1b (incorrect -> 0xa850), ack 93,
win 128, options [nop,nop,TS val 3834507601 ecr 305286128], length 0
E..4.t@.@..Z
o[D
o[...P..<Nb.....
...Q.2K.
09:14:20.964496 lxc4f4a182c887c In IP (tos 0x0, ttl 64, id 33397, offset 0, flags [DF], proto TCP
(6), length 1003)
10.111.91.68.http > 10.111.91.211.54424: Flags [P.], cksum 0xcfd2 (incorrect -> 0x44eb), seq
1:952, ack 93, win 128, options [nop,nop,TS val 3834507605 ecr 305286128], length 951: HTTP,
length: 951
HTTP/1.1 200 OK
server: istio-envoy
date: Fri, 24 Jan 2025 09:14:20 GMT
content-type: text/html
content-length: 615
last-modified: Wed, 14 Aug 2024 05:42:07 GMT
etag: "66bc43af-267"
accept-ranges: bytes
x-envoy-upstream-service-time: 0
connection: close
x-envoy-decorator-operation: webserver.test-istio-mtls.svc.cluster.local:80/*

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

Дополнительные ресурсы

- nginx.com
- nginx.org
- Документация модуля `istio`

Настройка визуализации сетевого стека кластера

Чтобы визуализировать сетевые взаимодействия в кластере — между подами, сервисами и внешними ресурсами — а также отслеживать активность и выявлять проблемы с сетью, используйте модуль `cilium-hubble`.

После включения модуля визуализация сетевого стека доступна в веб-интерфейсе Hubble.

Дополнительные ресурсы

- Документация модуля `cilium-hubble`

Переключение CNI с Flannel или Simple bridge на Cilium



Инструкция актуальна для Deckhouse Kubernetes Platform версии 1.75 и ниже.
Для DKP версии 1.76 и выше используйте руководство «Переключение CNI в кластере».



Миграция выполняется с полным простоем кластера: будут перезапущены все поды и узлы. Оценочное время выполнения — **около 1 часа**.

Для миграции выполните следующие шаги:

1. Включите модуль `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  settings:
    tunnelMode: VXLAN
    enabled: true
```

2. Дождитесь запуска компонентов Cilium:

```
d8 k -n d8-cni-cilium get pods
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
agent-5zzfv	2/2	Running	5 (23m ago)	26m
agent-gqb2b	2/2	Running	5 (23m ago)	26m
agent-wtv4p	2/2	Running	5 (23m ago)	26m
operator-856d69fd49-mlglv	2/2	Running	0	26m
safe-agent-updater-26qpk	3/3	Running	0	26m
safe-agent-updater-qlbrh	3/3	Running	0	26m
safe-agent-updater-wjlr5	3/3	Running	0	26m

Если поды агента Cilium не переходят в состояние `Ready`:

- Сохраните манифест `d8-kube-proxy`:

```
d8 k -n kube-system get ds d8-kube-proxy -oyaml > d8-kube-proxy.yaml
```

- Выполните действия для восстановления запуска:


```
d8 k delete validatingadmissionpolicies.admissionregistration.k8s.io label-objects.deckhouse.io
d8 k -n kube-system delete ds d8-kube-proxy
d8 k -n d8-cni-cilium delete po -l app=agent
```

- Если запуск Cilium продолжается более 5 минут, перезапустите `kube-proxy`:

```
d8 k -n kube-system delete pod -l k8s-app=kube-proxy
```

3. Сохраните манифест DaemonSet:

```
d8 k -n d8-cni-simple-bridge get ds simple-bridge -O yaml > simple-bridge.yaml
# или
d8 k -n d8-cni-flannel get ds flannel -O yaml > flannel.yaml
```

4. Отключите модуль `cni-simple-bridge` или `cni-flannel`. Выполняйте шаг после того, как поды Cilium перешли в состояние `Ready`:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-simple-bridge
# или
d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-flannel
```

5. Удалите `validating webhook` (если присутствует):

```
d8 k delete validatingwebhookconfigurations.admissionregistration.k8s.io d8-deckhouse-validating-webhook-handler-hooks
```

6. Удалите неймспейс старого CNI (если не удалился автоматически):

```
d8 k delete ns d8-cni-simple-bridge
# или
d8 k delete ns d8-cni-flannel
```

7. (Опционально) Очистите артефакты старого CNI. При необходимости удалите интерфейс и сбросьте правила iptables:

```
ip link delete cni0

iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -t raw -F
iptables -t raw -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
```

8. Перезапустите все поды:

```
d8 k delete pods -A --all --wait=false
```

9. Настройте внутренние балансировщики LoadBalancer (при использовании). Для всех TargetGroup, входящих во внутренний балансировщик, установите параметр:

```
Preserve client IP addresses = Off
```

10. Перезагрузите узлы кластера по одному, включая master-узлы.

Балансировка входящего трафика

В этом разделе описываются подходы к балансировке входящего трафика в Deckhouse Kubernetes Platform (DKP):

- NLB (Network Load Balancer) — работает на сетевом уровне, маршрутизирует трафик по IP-адресам и портам без анализа содержимого запросов.
- ALB (Application Load Balancer) — действует на прикладном уровне, анализирует HTTP(S)-заголовки, пути и домены. Поддерживает SSL-терминацию и маршрутизацию в зависимости от содержимого запроса.

Балансировка на сетевом уровне (NLB)

Балансировка NLB может быть организована двумя способами:

- с помощью внешнего балансировщика от облачного провайдера;
- средствами внутреннего балансировщика `metallb`, работающего как в облачных, так и в bare-metal-кластерах.

Балансировка на прикладном уровне (ALB)

Для балансировки трафика на уровне приложений в DKP доступны следующие решения:

- Ingress NGINX Controller (модуль `ingress-nginx`);
- Kubernetes Gateway API (модуль `alb`);
- Istio (модуль `istio`).

Отличие Kubernetes Gateway API от API Gateway

Kubernetes Gateway API и API Gateway выполняют разные функции:

- Kubernetes Gateway API — спецификация Kubernetes, определяющая набор ресурсов для настройки маршрутизации входящего трафика к приложениям.
- API Gateway — архитектурный компонент или программный продукт, предоставляющий единую точку входа к API приложений и централизованно реализующий такие функции, как аутентификация, авторизация и ограничение частоты запросов.

Иными словами, Kubernetes Gateway API описывает, как настроить маршрутизацию трафика, а API Gateway — это разновидность инфраструктуры, которая этот трафик обрабатывает. Некоторые API Gateway можно настраивать через Kubernetes Gateway API. Модуль `alb` является реализацией Kubernetes Gateway API.

Разделение ролей в модели Gateway API

При использовании модуля `alb` ответственность за настройку и публикацию приложений разделяется между следующими ролями:

- администратор кластера — разворачивает кластерную инфраструктуру шлюза через `ClusterALBInstance`;
- администратор неймспейса — разворачивает локальную инфраструктуру шлюза через `ALBInstance` и настраивает приём трафика через `ListenerSet` (hostname, TLS, порты);
- разработчики приложения — настраивают маршрутизацию к приложениям с помощью `HTTPRoute` и других ресурсов маршрутизации.

В типовом сценарии с общекластерным шлюзом `ListenerSet` создаёт администратор неймспейса, а `HTTPRoute` — разработчики приложения. Один и тот же человек может выполнять обе роли, если у него есть соответствующие права.

Как выбрать реализацию ALB

В таблице ниже приведены критерии выбора реализации ALB.

Критерий	Ingress NGINX	Gateway API	Istio
API публикации	Ingress + аннотации	Gateway, ListenerSet, маршруты	Gateway, VirtualService, DestinationRule
Протоколы	HTTP/HTTPS, gRPC (Ingress)	HTTP/HTTPS, gRPC, TLS, TCP, UDP	HTTP/HTTPS, gRPC, TCP (Istio Gateway)

Критерий	Ingress NGINX	Gateway API	Istio
Разделение ролей	IngressClass + Ingress	ClusterALBInstance/ALB Instance → ListenerSet → маршруты	IngressIstioController + Istio Gateway
Service mesh	Нет	Опционально (sidecar на прокси шлюза)	Да
Типичный сценарий	Классический Ingress, минимальные изменения	Новые приложения, мультитенантность, постепенная миграция с Ingress	Canary, mTLS, трассировка, продвинутая маршрутизация

Сравнение возможностей модулей ingress-nginx и alb

Оба модуля решают одну задачу — приём и маршрутизацию внешнего трафика к приложениям, но опираются на разные стандарты: ingress-nginx использует Ingress API с аннотациями, а alb — Kubernetes Gateway API. Модули можно использовать в кластере одновременно. В таблице ниже сравниваются их возможности в текущих версиях.

Служебные домены (веб-интерфейсы компонентов DKP и модулей по publicDomainTemplate) и домены приложений (маршруты разработчиков приложения) настраиваются по-разному. Подробности — в разделе «Публикация служебных доменов».

Параметр	ingress-nginx	alb
Стандарт маршрутизации	Ingress API с аннотациями	Kubernetes Gateway API
Реализация прокси	nginx	Envoy Proxy
Стадия жизненного цикла	General Availability	Preview
Развитие	Режим сопровождения: upstream-проект Ingress NGINX не развивает новые возможности, обновления безопасности предоставляет DKP	Активно развивается
Минимальная версия DKP	Доступен во всех поддерживаемых версиях	1.76
Редакции DKP	Все редакции	Все редакции
Модель разграничения ролей	Администратор кластера, администратор неймспейса	Администратор кластера, администратор неймспейса, разработчики приложения
Несколько независимых точек входа	Несколько Ingress-контроллеров, выбор через ingressClass	Несколько объектов Gateway, выбор через gatewayName; общекластерные шлюзы и шлюзы в неймспейсе
HTTP/HTTPS (HTTP/1.1, HTTP/2, HTTP/3)	Есть	Есть (включение HTTP/3)
WebSocket	Есть	Есть
gRPC	Есть	Есть
FastCGI	Есть	Нет
TCP	Нет	Есть (TCPRoute)
UDP	Нет	Есть (UDPRoute)
TLS passthrough	Есть	Есть (TLSRoute)
Proxy Protocol	Есть	Есть
Способы приёма трафика	Инлеты LoadBalancer, HostPort и HostWithFailover	Инлеты LoadBalancer и HostPort
Автоматический выпуск TLS-сертификатов (cert-manager)	Есть	Есть

Параметр	ingress-nginx	alb
Настройка политик HTTPS (версии TLS, шифры, HSTS)	Есть	По умолчанию TLSv1.2/1.3; HSTS — через аннотацию заголовков ответа
WAF	ModSecurity на уровне контроллера или Ingress-ресурса	ModSecurity/Coraza на уровне маршрута, набор правил OWASP CRS
Внешняя аутентификация	Есть	Есть
Ограничение доступа по IP (whitelist)	Есть	Есть
Базовая аутентификация	Есть	Есть
Ограничение частоты запросов	Есть	Есть
Закрепление сессии (session affinity)	Есть	Есть
GeoIP	Гео-статистика запросов в метриках	Добавление полей GeoIP в заголовки на основе баз MaxMind
Метрики Prometheus и дашборды Grafana	Есть, с детализацией по неймспейсам, виртуальным хостам, Ingress-ресурсам и локациям	Есть: метрики Envoy Proxy и дашборды по запросам, маршрутам и upstream
Трассировка OpenTelemetry	Есть	Есть (настройка)

Следующие шаги

Чтобы опубликовать приложение, выполните следующие шаги:

1. Выберите реализацию ALB по критериям из таблицы выше: Ingress NGINX, Gateway API или Istio.
 - Istio — если нужно управление трафиком в service mesh (canary-маршрутизация, mTLS между подами).
 - Gateway API — если нужна модель с разделением ролей и протоколами шире классического Ingress.
 - Ingress NGINX — если нужен зрелый ALB на Ingress API.
2. Включите и настройте соответствующий модуль. Для Gateway API выполните «Действия перед включением и настройкой ALB в кластере».
3. Опубликуйте приложения по руководствам пользователя по ALB (Gateway API, Ingress NGINX или Istio).
4. Для перехода с Ingress NGINX на Gateway API следуйте разделу «Миграция с ingress-nginx на alb».

Дополнительные ресурсы

- ALB средствами Ingress NGINX Controller
- ALB средствами Kubernetes Gateway API
- Миграция с ingress-nginx на alb
- ALB средствами Istio
- Использование Application Load Balancer (ALB)

Балансировка средствами MetalLB

Модуль `metallb` реализует поддержку сервисов типа LoadBalancer в кластерах Deckhouse Kubernetes Platform (DKP). Он подходит как для bare-metal-кластеров, так и для облачных, в которых недоступны встроенные балансировщики от провайдера.

Поддерживаются два режима работы:

- **Layer 2** — усовершенствованный (относительно стандартного режима L2 в MetalLB) механизм балансировки, который позволяет использовать несколько публичных адресов для сервисов.
- **BGP** — полностью основан на решении MetalLB и доступен только в DKP Enterprise Edition.

Режим Layer 2

Принцип работы

В режиме Layer 2 один или несколько узлов кластера принимают трафик к сервису из публичной сети. Сетевой уровень воспринимает это так, как будто у каждого из этих узлов назначено несколько IP-адресов на сетевом интерфейсе. Технически это реализуется следующим образом: модуль отвечает на ARP-запросы (для IPv4) и NDP-запросы (для IPv6).

Главное преимущество режима Layer 2 — универсальность: он работает в любой Ethernet-сети и не требует специализированного оборудования.

Преимущества модуля перед классическим MetalLB

В классическом MetalLB (режим L2) при создании сервиса с типом LoadBalancer балансировка работает за счёт того, что один из узлов кластера имитирует ARP-ответы от публичного IP. Это означает:

- Одновременно только один узел обслуживает весь входящий трафик для данного IP.
- Узел, выбранный в качестве лидера, становится «узким местом» без возможности горизонтального масштабирования.
- При отказе узла все активные соединения обрываются во время переключения на новый узел.

Модуль `metallb` устраняет эти ограничения. Он предоставляет ресурс `MetalLoadBalancerClass`, который позволяет:

- связать группу узлов с пулом IP-адресов с помощью `nodeSelector`;
- создать стандартный объект `Service` с типом `LoadBalancer` и указать имя нужного `MetalLoadBalancerClass`;
- через аннотацию определить количество IP-адресов для L2-анонсирования.

Таким образом:

- Приложение получает сразу несколько публичных IP. Их нужно указать как A-записи в DNS.
- Для масштабирования достаточно добавить балансировочные узлы; связанные с ними объекты `Service` создадутся автоматически — потребуется лишь добавить их в список A-записей прикладного домена.
- При выходе одного из балансировщиков из строя трафик переключится только частично, без полного разрыва соединений.

Сравнение поведения

Характеристика	Классический MetalLB (L2)	Новый модуль с <code>MetalLoadBalancerClass</code>
Обработка трафика	Один узел (лидер)	Несколько узлов
Масштабируемость	Нет	Да
Устойчивость к сбоям	Обрыв всех соединений	Часть трафика переключается плавно
Количество публичных IP	Один	Несколько (настраивается)
DNS-настройка	Одна A-запись	Несколько A-записей

Пример использования MetalLB в режиме L2 LoadBalancer

1. Включите модуль `metallb`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

2. Подготовьте приложение, которое хотите опубликовать:

```
d8 k create deploy nginx --image=nginx
```

3. Создайте ресурс MetalLoadBalancerClass:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: "" # Селектор узлов-балансировщиков.
  type: L2

```

4. Создайте ресурс Service с аннотацией и именем MetalLoadBalancerClass:

i При создании ресурса Service можно вообще не указывать аннотации `network.deckhouse.io/l2-load-balancer-external-ips-count` и `network.deckhouse.io/load-balancer-ips` из примеров ниже. В таком случае сервису автоматически выделится 1 случайный адрес из пула.

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # Имя MetalLoadBalancerClass.
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx

```

Если нужно присвоить сервису определенные IP-адреса из пула...

Если сервису необходимо выделить определенные IP-адреса из пула, используйте аннотацию `network.deckhouse.io/load-balancer-ips`. Если желаемых адресов больше одного, то также должна присутствовать аннотация `network.deckhouse.io/l2-load-balancer-external-ips-count`, в которой необходимо указать количество выделяемых адресов из пула (оно не должно быть меньше количества адресов, перечисленных в `network.deckhouse.io/load-balancer-ips`).

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # Количество адресов, которые будут выделены из пула, объявленного в
    MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # Список адресов из пула, объявленного в MetalLoadBalancerClass, которые будут
    выделены сервису.
    network.deckhouse.io/load-balancer-ips:
"192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # Имя MetalLoadBalancerClass.
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx

```

В результате, созданному сервису с типом LoadBalancer будут присвоены адреса в заданном количестве:

```
d8 k get svc
```

Пример вывода:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
nginx-deployment	LoadBalancer	10.222.130.11	
192.168.2.100,192.168.2.101,192.168.2.102		80:30544/TCP	11s

Полученные **EXTERNAL-IP** можно прописывать в качестве А-записей для прикладного домена:

```

curl -S -O /dev/null -w "%{http_code}" 192.168.2.100:8000
curl -S -O /dev/null -w "%{http_code}" 192.168.2.101:8000
curl -S -O /dev/null -w "%{http_code}" 192.168.2.102:8000

```

Пример вывода:

```
200
```

Режим BGP

i Доступен только в DKP Enterprise Edition.

metallb в режиме BGP используется для предоставления сервисов типа LoadBalancer в кластерах Kubernetes, развёрнутых на физической инфраструктуре. IP-адреса сервисов анонсируются напрямую в маршрутизаторы (или Top-of-Rack-коммутаторы) с помощью протокола BGP.

Принцип работы

Конфигурация

- Определяется пул IP-адресов, доступных для назначения сервисам.
- Указываются параметры BGP-сессий: AS (Autonomous System) number кластера Kubernetes, IP-адреса маршрутизаторов (пиров), AS number пиров, пароли для аутентификации (при

необходимости).

- Для каждого пула IP-адресов могут быть заданы специфические параметры анонсирования, например, `community strings`.

Установка BGP-сессий

- На каждом узле кластера Kubernetes, где запущен `metallb`, компонент `speaker` устанавливает BGP-сессии с указанными маршрутизаторами.
- Выполняется обмен маршрутной информацией между `metallb` и маршрутизаторами.

Назначение IP-адресов сервисам

- При создании объекта Service типа LoadBalancer, `metallb` выбирает свободный IP-адрес из сконфигурированного пула и назначает его сервису.
- Компонент `controller` отслеживает изменения в сервисах и управляет назначениями IP-адресов.

Анонсирование IP-адресов

- После назначения IP-адреса компонент `speaker` на одном из узлов (лидере для данного сервиса) начинает анонсировать его через установленные BGP-сессии.
- Маршрутизаторы получают анонс и обновляют таблицы маршрутизации, направляя трафик на соответствующий узел.

Распределение трафика

- Маршрутизаторы используют протоколы Equal-Cost Multi-Path (ECMP) или другие алгоритмы балансировки для распределения трафика между узлами, анонсирующими один и тот же IP-адрес сервиса.
- После доставки на узел входящий трафик перенаправляется на поды сервиса с помощью механизмов используемого CNI (iptables/IPVS, eBPF-программы и т.д.).

Преимущества использования BGP

- Протокол BGP поддерживается большинством сетевого оборудования.
- Сеть может включать несколько маршрутизаторов и большое число узлов.
- Балансировка трафика через ECMP.
- При выходе из строя узла, анонсирующего IP-адрес, маршрутизаторы автоматически перенаправляют трафик на другие узлы с этим же IP.

Недостатки использования BGP

- Более высокая сложность настройки, чем настройка анонсов ARP/GARP (в режиме Layer 2).
- Маршрутизаторы должны поддерживать BGP и ECMP.

Пример использования MetalLB в режиме BGP LoadBalancer

1. Включите модуль `metallb` и настройте все необходимые параметры:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  settings:
    addressPools:
      - addresses:
          - 192.168.219.100-192.168.219.200
        name: mypool
        protocol: bgp
    bgpPeers:
      - hold-time: 3s
        my-asn: 64600
        peer-address: 172.18.18.10
        peer-asn: 64601
    speaker:
      nodeSelector:
        node-role.deckhouse.io/metallb: ""
  version: 2

```

2. Настройте BGP-пиринг на сетевом оборудовании.

Дополнительные ресурсы

- Документация модуля `metallb`

Балансировка в кластерах на облачных платформах

Перечень поддерживаемых провайдеров:

- Amazon Web Services,
- Google Cloud Platform,
- Microsoft Azure,
- OpenStack,
- Huawei Cloud,
- VMware Cloud DirectorExperimental,
- VMware vSphere,
- Yandex Cloud,
- zVirtExperimental,
- Basis Dynamix.

Настройка балансировки входящего трафика в кластерах на облачных платформах включает в себя создание Ingress-контроллера с указанием параметров LoadBalancer. На их основе облачный провайдер автоматически создаёт внешний балансировщик нагрузки. В кластере при этом создаётся ресурс Service, через который трафик от внешнего балансировщика будет направляться к приложениям.

Пример создания Ingress-контроллера для провайдера OpenStack

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend

```

Пример создания сервиса с типом ClusterIP

```

apiVersion: v1
kind: Service
metadata:
  name: backend-resolver-cluster-ip
spec:
  ports:
    - name: http
      port: 8000
      protocol: TCP
  selector:
    app: lab-4-backend
  type: ClusterIP

```

Пример создания внутреннего балансировщика для VK Cloud

Этот пример подходит, когда нужно создать балансировщик только внутри сети облака (без внешнего адреса).

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker

```

ALB средствами Ingress NGINX Controller

Для реализации ALB средствами Ingress NGINX Controller используется модуль `ingress-nginx`.

i В 2025 году Ingress NGINX был переведён в режим сопровождения без планов активного развития новых возможностей. Дальнейшее развитие средств балансировки входящего трафика в Kubernetes ориентировано на Gateway API.

На поддержку модуля в составе Deckhouse Kubernetes Platform (DKP) это не распространяется: модуль сопровождается командой DKP, включая обновления безопасности. Подробности — в разделе «Поддержка и безопасность модуля».

Пошаговый переход на Gateway API — в разделе «Миграция с ingress-nginx на alb».

Модуль `ingress-nginx` устанавливает Ingress NGINX Controller и управляет им с помощью кастомных ресурсов. Если узлов для размещения Ingress-контроллера больше одного, он устанавливается в отказоустойчивом режиме, с учётом особенностей инфраструктуры как облачных, так и bare-metal-сред, а также различных типов Kubernetes-кластеров.

Поддерживается одновременный запуск нескольких экземпляров Ingress-контроллеров с независимой конфигурацией: одного основного и произвольного количества дополнительных. Это, например, позволяет разделять внешние и внутренние (intranet) Ingress-ресурсы приложений.

Варианты терминации трафика

Трафик к `ingress-nginx` может быть отправлен несколькими способами:

- напрямую, без использования внешнего балансировщика;
- через внешний LoadBalancer, в том числе поддерживаются:
 - Qrator,
 - Cloudflare,
 - AWS LB,
 - GCE LB,
 - ACS LB,
 - Yandex LB,
 - OpenStack LB.

Терминация HTTPS

Для каждого экземпляра Ingress NGINX Controller можно настраивать политики безопасности HTTPS, включая:

- параметры HSTS;
- набор доступных версий SSL/TLS и протоколов шифрования.

Также модуль интегрирован с модулем `cert-manager`, при взаимодействии с которым возможны автоматический заказ SSL-сертификатов и их дальнейшее использование Ingress-контроллерами.

Мониторинг и статистика

В этой реализации `ingress-nginx` добавлена система сбора статистики в Prometheus со следующим набором метрик:

- длительность всего ответа и ответа бэкенда отдельно;
- коды ответа;
- количество повторов запросов (retry);
- размеры запроса и ответа;
- методы запросов;
- типы `content-type`;
- география распределения запросов и т. д.

Данные представлены в нескольких разрезах:

- `namespace`;
- `vhost`;
- `ingress-ресурсы`;
- `location` (в `nginx`).

Все графики сгруппированы в дашборды Grafana. С любого графика можно перейти к более детальному представлению: например, при просмотре статистики по `namespace` — на дашборд по `vhosts` в этом `namespace` и далее по иерархии.

Статистика

Ниже описаны принципы сбора статистики и состав собираемых метрик.

Основные принципы сбора статистики

Сбор статистики устроен следующим образом:

1. На стадии `log_by_lua_block` для каждого запроса вызывается модуль, который рассчитывает необходимые данные и складывает их в буфер (у каждого NGINX worker свой буфер).
2. На стадии `init_by_lua_block` для каждого NGINX worker запускается процесс, который раз в секунду асинхронно отправляет данные в формате `protobuf` через TCP socket в `protobuf_exporter` (разработка Deckhouse Kubernetes Platform).
3. `protobuf_exporter` запускается sidecar-контейнером в поде с `ingress-controller`, принимает сообщения в формате `protobuf`, разбирает, агрегирует их по установленным нами правилам и экспортирует в формате для Prometheus.
4. Prometheus каждые 30 секунд собирает метрики как в `ingress-controller` (там есть небольшое количество нужных метрик), так и `protobuf_exporter`. На основе этих данных строится статистика.

Состав и представление метрик

У всех собираемых метрик есть служебные лейблы, идентифицирующие экземпляр контроллера: `controller`, `app`, `instance` и `endpoint` (они видны в `/prometheus/targets`).

- Все метрики (кроме `geo`), экспортируемые через `protobuf_exporter`, представлены в трёх уровнях детализации:
 - `ingress_nginx_overall_*` — агрегированные метрики верхнего уровня (без детализации, у всех метрик есть лейблы `namespace`, `vhost` и `content_kind`);
 - `ingress_nginx_detail_*` — кроме лейблов уровня `overall`, добавляются `ingress`, `service`, `service_port` и `location`;
 - `ingress_nginx_detail_backend_*` — ограниченная часть данных, собирается в разрезе по бэкендам. У этих метрик, кроме лейблов уровня `detail`, добавляется лейбл `pod_ip`.
- Для уровней `overall` и `detail` собираются следующие метрики:
 - `*_requests_total` — общее количество запросов (дополнительные лейблы — `scheme`, `method`);
 - `*_responses_total` — количество ответов (дополнительный лейбл — `status`);
 - `*_request_seconds {sum,count,bucket}` — гистограмма времени ответа;
 - `*_bytes_received {sum,count,bucket}` — гистограмма размера запроса;
 - `*_bytes_sent {sum,count,bucket}` — гистограмма размера ответа;
 - `*_upstream_response_seconds {sum,count,bucket}` — гистограмма времени ответа upstream-сервиса (при нескольких upstream'ах — суммарное время);
 - `*_lowres_upstream_response_seconds {sum,count,bucket}` — упрощённая гистограмма (для визуализации; не подходит для расчёта квантилей);
 - `*_upstream_retries {count,sum}` — количество и суммарное число повторных запросов (`retry`) к бэкенду.
- Для уровня `overall` собираются следующие метрики:
 - `*_geohash_total` — количество запросов по `geohash` (дополнительные лейблы — `geohash`, `place`).
- Для уровня `detail backend` собираются следующие метрики:
 - `*_lowres_upstream_response_seconds` — упрощённая гистограмма времени ответа для `overall` и `detail`;
 - `*_responses_total` — количество ответов (дополнительный лейбл — `status_class`, а не просто `status`);
 - `*_upstream_bytes_received_sum` — суммарный объём данных, полученных от бэкендов.

Примеры настройки балансировки

Для настройки балансировки используйте кастомный ресурс `IngressNginxController`.

- AWS (NLB)
- GCP, Yandex Cloud и Azure
- OpenStack
- VK Cloud
- Bare metal (HostWithFailover)
- Bare metal с внешним LB
- MetalLB BGP
- MetalLB L2

Пример для AWS (Network Load Balancer)

При создании балансировщика используются все доступные в кластере зоны.

В каждой зоне балансировщик получает собственный публичный IP. Если в зоне есть экземпляр с Ingress-контроллером, A-запись с IP-адресом балансировщика из этой зоны автоматически добавляется к доменному имени балансировщика.

Если в зоне не остаётся экземпляров с Ingress-контроллером, тогда IP автоматически убирается из DNS.

В том случае, если в зоне всего один экземпляр с Ingress-контроллером, при перезапуске пода IP-адрес балансировщика этой зоны временно исключается из DNS.

IngressNginxController с инлетом `LoadBalancer` и аннотациями AWS NLB:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

Пример для GCP, Yandex Cloud и Azure

IngressNginxController с инлетом `LoadBalancer`:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
```

i В GCP на узлах необходимо указать аннотацию, которая разрешает принимать подключения на внешние адреса для объектов Service с типом NodePort.

Пример для OpenStack

IngressNginxController с инлетом `LoadBalancerWithProxyProtocol` и аннотациями Proxy Protocol для OpenStack:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main-lbwpp
spec:
  inlet: LoadBalancerWithProxyProtocol
  ingressClass: nginx
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
```

Пример создания внутреннего балансировщика для VK Cloud

Подходит, когда нужно создать балансировщик только внутри сети облака (без внешнего адреса).

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker

```

Пример для bare metal

IngressNginxController с инлетом `HostWithFailover` на frontend-узлах:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostWithFailover
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend

```

Пример для bare metal при использовании внешнего балансировщика

Подходит при использовании Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp и других внешних балансировщиков.

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true

```

Пример для bare metal (MetalLB в режиме BGP LoadBalancer)

i Доступно только в DKP Enterprise Edition.

IngressNginxController с инлетом `LoadBalancer` для использования с MetalLB в режиме BGP:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend

```

В случае использования MetalLB его speaker-поды (компонент MetalLB, анонсирующий IP-адреса) должны быть запущены на тех же узлах, что и поды Ingress-контроллера.

Чтобы Ingress-контроллер получал реальные IP-адреса клиентов, его Service должен быть создан с параметром `externalTrafficPolicy: Local`, исключающим межузловой SNAT. MetalLB speaker анонсирует этот Service только с узлов, где запущены целевые поды.

Таким образом, для данного примера конфигурация модуля `metallb` через ModuleConfig должна быть такой:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
  settings:
    speaker:
      nodeSelector:
        node-role.deckhouse.io/frontend: ""
      tolerations:
        - effect: NoExecute
          key: dedicated.deckhouse.io
          value: frontend

```

Пример для bare metal (балансировщик MetalLB в режиме L2 LoadBalancer)

i Доступно только в DKP Enterprise Edition.

1. Включите модуль `metallb`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2

```

2. Создайте ресурс MetalLoadBalancerClass:

i MetalLB-балансировщики должны размещаться на тех же узлах, что и Ingress-контроллеры. В типовых сценариях развёртывания для этой цели используются frontend-узлы.

Чтобы разместить Ingress-контроллеры и MetalLB-балансировщики на frontend-узлах, укажите в `nodeSelector` лейбл `node-role.deckhouse.io/frontend: ""`.

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.deckhouse.io/frontend: "" # Селектор узлов-балансировщиков.
  type: L2
```

3. Создайте ресурс IngressNginxController:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    loadBalancerClass: ingress
  annotations:
    # Количество адресов, которые будут выделены из пула, описанного в
    MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
      operator: Equal
```

i При создании Ingress-контроллера можно явно указать IP-адреса из пула, которые должны быть назначены его Service. Для этого используйте аннотацию `network.deckhouse.io/load-balancer-ips`.

Если требуется назначить несколько IP-адресов, дополнительно укажите аннотацию `network.deckhouse.io/l2-load-balancer-external-ips-count` с количеством выделяемых из пула адресов. Указанное значение не должно быть меньше количества IP-адресов, перечисленных в `network.deckhouse.io/load-balancer-ips`.

Пример настройки приведён в разделе «Создание сервиса с присвоением ему определённых IP-адресов из пула».

DKP создаст Service с типом LoadBalancer, которому будет присвоено заданное количество адресов:

```
d8 k -n d8-ingress-nginx get svc
```

Пример вывода:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S)	AGE		
main-load-balancer	LoadBalancer	10.222.130.11	
192.168.2.100, 192.168.2.101, 192.168.2.102		80:30689/TCP, 443:30668/TCP	11s

Пример разделения доступа между публичной и административной зонами

Во многих приложениях один и тот же бэкенд обслуживает как публичную часть, так и административный интерфейс. Например:

- `https://example.com` — публичная зона;
- `https://admin.example.com` — административная зона, к которой доступ должен быть ограничен (ACL, mTLS, IP whitelist и так далее).

При таком сценарии административный трафик рекомендуется выносить в отдельный Ingress-контроллер (при необходимости с отдельным Ingress-классом) и ограничивать доступ к нему с помощью параметра `spec.acceptRequestsFrom`.

- Один Ingress-контроллер
- Раздельные Ingress-контроллеры

Особенности использования одного Ingress-контроллера

Ниже показан пример, когда для обслуживания запросов из публичной зоны и административного интерфейса используется один Ingress-контроллер.

Пример конфигурации Ingress-ресурсов для такого случая:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: nginx # Ingress-ресурс для административного трафика связан с тем
  же Ingress-контроллером, что и Ingress-ресурс для публичного трафика.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: nginx # Ingress-ресурс для публичного трафика связан с тем же
  Ingress-контроллером, что и Ingress-ресурс для административного трафика.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80

```

При включённой обработке и передаче заголовков `X-Forwarded-*` бэкенд может опираться на заголовок `x-forwarded-host` при принятии решений об авторизации. В примере выше через публичный Ingress можно подключиться к административной зоне с помощью `x-forwarded-host`. Поэтому запросы к Ingress-контроллеру должны приходить только от доверенных источников.

Использование отдельных Ingress-контроллеров

Чтобы избежать этой ситуации, выполните следующее:

- настроить правила доступа на уровне Ingress-ресурсов,
- использовать разные Ingress-контроллеры,
- на Ingress-контроллерах ограничить, с каких адресов можно подключаться к ним.

Пример конфигурации Ingress-ресурсов для такого случая:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: admin-nginx # Ingress-ресурс для административного трафика связан
  с отдельным Ingress-контроллером.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: public-nginx # Ingress-ресурс для публичного трафика связан с
  отдельным Ingress-контроллером.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80

```

Пример Ingress-контроллера, который обслуживает административные Ingress-ресурсы и принимает подключения только из заданных подсетей:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: admin
spec:
  ingressClass: admin-nginx
  inlet: HostPort
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true

```

В этом примере:

- Ingress-контроллер доступен на портах узлов через инлет `HostPort`.
- Параметр `acceptRequestsFrom` разрешает подключение к контроллеру только из

- перечисленных подсетей.
- Даже если внешний балансировщик или клиент может передавать свои значения заголовков `X-Forwarded-*`, решение о допуске соединения до контроллера принимается по реальному адресу подключения, а не по заголовкам.
- Административные Ingress-ресурсы (в данном примере `admin-ingress`) обслуживаются этим контроллером согласно настроенному Ingress-классу.

Пример Ingress-контроллера, который обслуживает Ingress-ресурсы для публичного трафика:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: public
spec:
  ingressClass: public-nginx
  inlet: HostPort
  hostPort:
    httpPort: 8080
    httpsPort: 8443
    behindL7Proxy: true
```

Поддержка и безопасность модуля

Модуль `ingress-nginx` входит в сопровождение DKP на весь срок поддержки платформы, вне зависимости от режима развития upstream-проекта. Команда DKP отслеживает CVE в контроллере и зависимостях — NGINX, Lua-модули, базовые образы — и предоставляет исправления в релизах платформы.

Для соответствия ожиданиям PCI DSS по вендорской поддержке и срокам устранения уязвимостей ответственным вендором модуля является компания «Флант». Сертификация DKP в ФСТЭК России фиксирует, в том числе, процессы управления уязвимостями и выпуск обновлений безопасности.

Дополнительные ресурсы

- Миграция с `ingress-nginx` на `alb`
- ALB средствами Kubernetes Gateway API
- Использование Application Load Balancer (ALB)
- Документация модуля `ingress-nginx`
- Документация модуля `metallb`

ALB средствами Kubernetes Gateway API

Для реализации прикладного балансировщика нагрузки (Application Load Balancer, ALB) средствами Kubernetes Gateway API используется модуль `alb`.

Модуль `alb` реализует прикладной балансировщик нагрузки и позволяет публиковать приложения с помощью Kubernetes Gateway API. Он разворачивает и настраивает инфраструктуру для приёма и маршрутизации внешних запросов, а также проверяет пользовательскую конфигурацию Gateway API.

i

ALB средствами Kubernetes Gateway API может использоваться в кластере совместно с ALB средствами Ingress NGINX Controller. О совместном использовании с другими модулями и сторонними решениями читайте в разделе «Совместное использование с другими модулями и сторонними решениями».

Обзор и схема

Модуль построен на Kubernetes Gateway API — API маршрутизации входящего трафика, расширяющий модель Ingress API. Объект `ClusterALBInstance` или `ALBInstance` создаёт управляемый Gateway. К Gateway привязывается `ListenerSet`, который описывает обработчики входящих запросов. Маршруты направляют трафик к сервисам приложений.

Модуль поддерживает:

- единый декларативный API для HTTP/HTTPS, gRPC, TCP, UDP и TLS passthrough;
- разделение ответственности между администратором кластера (ClusterALBInstance или ALBInstance), администратором неймспейса (ALBInstance и ListenerSet — hostname, TLS, порты) и разработчиками приложения (маршруты);
- обработку запросов: WAF на уровне маршрута, внешнюю аутентификацию, списки разрешённых IP-адресов, ограничение частоты запросов, закрепление сессии (session affinity), GeoIP, BackendTLSPolicy, Proxy Protocol и HTTP/3.

Kubernetes Gateway API и API Gateway — разные понятия. Kubernetes Gateway API — это набор ресурсов Kubernetes для описания маршрутизации трафика к приложениям. API Gateway — архитектурный компонент, предоставляющий единую точку входа к API приложений. Модуль `alb` реализует Kubernetes Gateway API.

Сравнение возможностей модулей `alb` и `ingress-nginx` приведено в разделе «Сравнение возможностей модулей `ingress-nginx` и `alb`».

Роли объектов

Gateway API разделяет ответственность между администраторами кластера и неймспейса и разработчиками приложений:

- Администратор кластера — управляет инфраструктурой приёма трафика через ClusterALBInstance или ALBInstance (общекластерный Gateway).
- Администратор неймспейса — управляет ALBInstance и ListenerSet (hostname, TLS, порты) в неймспейсе.
- Разработчики приложения — задают маршруты (HTTPRoute, GRPCRoute, TLSRoute, TCPRoute, UDPRoute).

Зачем нужен ListenerSet

ListenerSet — расширение Gateway API. Объект ListenerSet описывает системные и пользовательские обработчики трафика: имя хоста, режим TLS, порт и протокол. Каждый ListenerSet связывается с родительским Gateway через `spec.parentRef`. Затем к нему подключаются маршруты.

Каждый объект Gateway по умолчанию создаёт два обработчика: `d8-http` (порт 80) и `d8-https` (порт 443). Они предназначены для служебных целей — например, для проверки доступности шлюза или работы `cert-manager` (HTTP-01). Для публикации приложений эти обработчики использовать не рекомендуется. Используйте ListenerSet.

ClusterALBInstance и ALBInstance

При создании управляемого объекта Gateway для публикации пользовательских приложений используются кастомные ресурсы ClusterALBInstance (общекластерный объект) и ALBInstance (объект в неймспейсе).

Особенности и разница между ними описаны в таблице:

	ClusterALBInstance	ALBInstance
Назначение	Развёртывание общекластерного Gateway	Развёртывание Gateway в неймспейсе

	ClusterALBInstance	ALBInstance
Сценарии использования	- Общая точка входа (общекластерный шлюз). - Системный шлюз для публикации веб-интерфейсов служебных компонентов Deckhouse Kubernetes Platform (DKP) и других модулей (может требоваться «Действия перед включением и настройкой ALB в кластере»). - Платформенный шлюз	Отдельный шлюз для приложения или команды в выделенном неймспейсе
Поддерживаемые типы инлета	LoadBalancer, HostPort	LoadBalancer, ClusterIP
Реализация прокси	Envoy Proxy	Envoy Proxy
Тип развёртывания	DaemonSet	Deployment
Локализация объектов ListenerSet и маршрутов	В любом пользовательском неймспейсе	В том же неймспейсе, что и объект ALBInstance (обязательно)
Права доступа	Администратор кластера	Администратор неймспейса

После создания ClusterALBInstance или ALBInstance в кластере появляется управляемый объект Gateway (шлюз). При этом:

- Каждый объект Gateway обслуживается как минимум одним экземпляром Envoy Proxy.
- Трафик в него приходит через сервис с типом LoadBalancer или напрямую с использованием параметров HostPort.
- На один объект Gateway могут ссылаться несколько объектов ClusterALBInstance или ALBInstance через поле gatewayName. В этом случае они используют общий Gateway. Конфигурация инфраструктуры для приёма и обработки трафика при этом может различаться. Поле gatewayName можно рассматривать как аналог ingressClass для объектов IngressNginxController. Итоговая конфигурация берётся из инстанса с наиболее ранним creationTimestamp (то есть созданного раньше остальных). У остальных инстансов в статусе устанавливается conflictPorts: true, а в conflictPortsOwner указывается имя «управляющего» инстанса.

Валидация конфигурации

Помимо настройки инфраструктуры Gateway API выполняется проверка пользовательских настроек перед применением, чтобы не допустить конфликта. Это позволяет выявлять конфликты между одинаковыми обработчиками трафика в разных объектах ListenerSet, если они ссылаются на один и тот же объект Gateway.

Инлеты

Инлет определяет, как внешний трафик попадает на Envoy Proxy, обслуживающий объект Gateway:

- LoadBalancer — приём трафика через сервис с типом LoadBalancer (облачные провайдеры или bare metal с MetalLB). Доступен и для ClusterALBInstance, и для ALBInstance.
- HostPort — приём трафика на портах узлов без внешнего балансировщика. Доступен только для ClusterALBInstance.
- ClusterIP — приём трафика через сервис с типом ClusterIP, без внешней точки входа (например, для внутреннего трафика). Доступен только для ALBInstance.

Таблица соответствия типов инлета при миграции с ingress-nginx — в разделе «Настройка инлета». Подробные примеры — в разделе «Примеры для разных окружений».

Как настроить ALB

Публикация приложения предполагает включение модуля, создание управляемого Gateway, ListenerSet и маршрутов.

Действия перед включением и настройкой ALB в кластере

Перед включением и настройкой ALB в кластере DKP выполните следующее:

- Проверьте выполнение требований к работе модуля `alb`.
- Если нужно публиковать служебные домены — веб-интерфейсы служебных компонентов DKP и других модулей, — укажите глобальный параметр `publicDomainTemplate`. Без этого параметра системные объекты `HTTPRoute`, `Gateway` и `ListenerSet` для служебных доменов не будут работать корректно, и веб-интерфейсы не опубликуются. Если публикация служебных доменов не требуется, параметр можно не указывать. Подробности — в разделе «Публикация служебных доменов».
- Проверьте совместимость версий API в разделе «Совместно со сторонними реализациями Gateway API», если такие решения уже используются в кластере.
- На bare metal для инлета `LoadBalancer` подготовьте внешний балансировщик или модуль `metallb`. Инлет `HostPort` доступен только для `ClusterALBInstance` и не требует `MetalLB`.

Включение модуля и создание Gateway

Включите модуль `alb`, как описано в разделе «Включение модуля».

Создайте `ClusterALBInstance` или `ALBInstance`. Это приведёт к созданию и настройке управляемого объекта `Gateway`.



Ручная модификация объектов `Gateway`, управляемых модулем, не допускается.

- `ClusterALBInstance`
- `ALBInstance`

Пример манифеста `ClusterALBInstance` для создания общекластерного шлюза:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

Пример минимальной рабочей конфигурации `ALBInstance` для публикации шлюза в неймспейсе:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

После перехода `ALBInstance` в состояние `Ready` создайте в том же неймспейсе объекты `ListenerSet` и `HTTPRoute`. Порядок их настройки и пример публикации приложения приведены в разделе «Публикация приложения с `ListenerSet` и `HTTPRoute`».

Создание ListenerSet

Объект `ListenerSet` описывает системные и пользовательские обработчики трафика: имя хоста, режим TLS, порт и протокол. Каждый `ListenerSet` связывается с родительским `Gateway` через `spec.parentRef`. Затем к нему подключаются маршруты.

Расположение объектов `ListenerSet` зависит от используемого типа объекта `Gateway`:

- Для `ClusterALBInstance` объекты `ListenerSet` могут располагаться в любом неймспейсе.
- Для `ALBInstance` объекты `ListenerSet` должны располагаться в том же неймспейсе, что и родительский `ALBInstance`.

В обоих случаях рекомендуется размещать `ListenerSet` в одном неймспейсе со связанными

объектами HTTPRoute, GRPCRoute и TLSRoute. В этом случае дополнительные настройки, например создание ReferenceGrant, не требуются.

В ListenerSet для HTTP/HTTPS указывайте порты 80 и 443. Это порты слушателей Gateway API. Они не совпадают с параметрами `hostPort.httpPort` и `hostPort.httpsPort` инлета HostPort, которые задают порты на узле.

Объекты TCPRoute и UDPRoute, использующие TCP и UDP-порты из `additionalPorts`, привязываются непосредственно к соответствующему слушателю объекта Gateway.

Пример манифеста ListenerSet для управления приёмом входящих HTTP и HTTPS-запросов через общекластерный шлюз:

```
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: prod
spec:
  parentRef:
    name: public-gw # Имя общекластерного Gateway из status ClusterALBInstance.
    namespace: d8-alb
  listeners:
    - name: app-http
      port: 80 # В ListenerSet для HTTP указывайте порт 80. Это не параметр
      hostPort.httpPort инлета HostPort на узле.
      protocol: HTTP
      hostname: app.example.com
    - name: app-https
      port: 443 # В ListenerSet для HTTPS указывайте порт 443. Это не параметр
      hostPort.httpsPort инлета HostPort на узле.
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls # Наименование объекта Secret, содержащего необходимый TLS-
            сертификат.
            namespace: prod
```

Создание маршрутов

Для маршрутизации входящих запросов используются следующие типы маршрутов:

- HTTPRoute — для маршрутизации HTTP/HTTPS/TLS запросов. Объекты HTTPRoute поддерживают расширенные настройки с помощью аннотаций, которые дополняют текущую спецификацию Gateway API.
- GRPCRoute — для маршрутизации gRPC-трафика.
- TLSRoute — для сквозной маршрутизации TLS-трафика.
- TCPRoute — для маршрутизации TCP-трафика. Для TCP-портов из `additionalPorts` объект TCPRoute привязывается напрямую к слушателю Gateway, а не к ListenerSet.
- UDPRoute — для маршрутизации UDP-трафика. Для UDP-портов из `additionalPorts` объект UDPRoute привязывается напрямую к слушателю Gateway, а не к ListenerSet.

Объекты HTTPRoute, GRPCRoute и TLSRoute привязываются к ListenerSet. TCPRoute и UDPRoute для TCP/UDP-портов из `additionalPorts` привязываются напрямую к слушателю объекта Gateway.

Пример создания HTTPRoute для HTTP и HTTPS-трафика и другие сценарии публикации — в разделе «Публикация приложения с ListenerSet и HTTPRoute».

Публикация служебных доменов



Если нужно публиковать служебные домены, укажите глобальный параметр `publicDomainTemplate` — без него системные объекты `HTTPRoute`/`Gateway`/`ListenerSet` для служебных доменов не будут работать корректно, и веб-интерфейсы служебных компонентов DKP и других модулей не будут опубликованы. Если публикация служебных доменов не нужна, параметр можно не указывать.

Для предоставления доступа к служебным доменам кластера DKP укажите шлюз по умолчанию. Создайте `ClusterALBInstance` с нужным типом инлета и настройками и установите для него параметр `spec.defaultDeckhouseGateway: true`.

Пример манифеста `ClusterALBInstance` с параметром `spec.defaultDeckhouseGateway: true`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  defaultDeckhouseGateway: true
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

После применения изменений проверьте статус `ClusterALBInstance`:

```
d8 k get clusteralbinstances
```

`ClusterALBInstance` должен перейти в состояние `Ready` и создать управляемый Gateway. После этого в соответствующих системных неймспейсах появятся `ListenerSet` и `HTTPRoute`.



В настоящее время не все модули DKP доступны через Gateway API. Не отключайте модуль `ingress-nginx` и не удаляйте связанные с ним объекты, пока необходимые веб-интерфейсы не будут опубликованы через Gateway API и не пройдут проверку.

После настройки шлюза по умолчанию выполните следующую команду, чтобы увидеть, какие модули **уже** опубликовали служебные `HTTPRoute` через Gateway API в этом кластере. Команда показывает фактический список уже опубликованных маршрутов, а не полный перечень возможностей платформы:

```
d8 k get httproutes -A -l heritage=deckhouse -o json \
  | jq -r '[
    "MODULE", "GATEWAY API SUPPORT",
    (.items
      | map(.metadata.labels.module // "UNKNOWN")
      | unique[]
      | ["d8-" + ., "yes"])
    ]' | @tsv
  | column -t -s $'\t'
```

Выбор шлюза DKP по умолчанию при использовании нескольких ClusterALBInstance

В кластере может быть одновременно несколько общекластерных Gateway, помеченных как шлюз по умолчанию (флаг `spec.defaultDeckhouseGateway: true` в параметрах соответствующих `ClusterALBInstance`). В этом случае шлюзом по умолчанию становится Gateway, созданный объектом `ClusterALBInstance` с наиболее ранним `creationTimestamp` (то есть созданный раньше остальных).

Если ни один объект `ClusterALBInstance` не отмечен как шлюз по умолчанию, DKP допускает использование объекта Gateway, созданного модулем `alb` для инстанса `ClusterALBInstance` с именем `main`.

Смена шлюза DKP по умолчанию

Если системные домены DKP необходимо перевести на обслуживание другим объектом Gateway, выполните следующие шаги:

1. Создайте новый объект `ClusterALBInstance`, описывающий необходимые настройки, и задайте в нём параметр `spec.defaultDeckhouseGateway: true`.
2. В текущем объекте `ClusterALBInstance`, который предоставляет шлюз DKP по умолчанию, задайте `spec.defaultDeckhouseGateway: false`.
3. Проверьте, что все системные объекты `ListenerSet` теперь ссылаются на новый объект Gateway в `spec.parentRef`.

Примеры для разных окружений

В этом разделе описаны приём трафика в разных окружениях и операции с уже развёрнутым ALB. Полное описание полей кастомных ресурсов приведено в документации `ClusterALBInstance` и `ALBInstance`. Дополнительные примеры — в примерах модуля `alb`.

- Облачный провайдер
- Bare metal с MetalLB
- Bare metal с HostPort
- Внешний L7-балансировщик

Пример `ClusterALBInstance` с инлетом `LoadBalancer` для облачного провайдера:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

Чтобы настроить облачный балансировщик, укажите необходимые аннотации для создаваемого объекта `Service` типа `LoadBalancer` в параметре `spec.inlet.loadBalancer.serviceAnnotations`. Пример использования `Network Load Balancer (NLB)` в AWS:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer:
      serviceAnnotations:
        service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

Чтобы принимать трафик на bare metal с помощью балансировщика `MetalLB`, включите модуль `metallb` и создайте объект `MetalLoadBalancerClass` с пулом адресов. Разместите балансировщики `MetalLB` на тех же узлах, что и поды `Envoy Proxy` модуля `alb` (обычно `frontend`-узлы с лейблом `node-role.deckhouse.io/frontend`):

```

apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: alb
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  type: L2

```

Затем создайте объект `ClusterALBInstance` и укажите `spec.inlet.loadBalancer.loadBalancerClass`:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer:
      loadBalancerClass: alb
    serviceAnnotations:
      # Число адресов, выделяемых из пула MetalLoadBalancerClass.
      network.deckhouse.io/l2-load-balancer-external-ips-count: "1"

```

Пример `ClusterALBInstance` с инлетом `HostPort` для bare metal без внешнего балансировщика:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: HostPort
    hostPort:
      httpPort: 80
      httpsPort: 443

```

Чтобы поды Envoy Proxu размещались только на выделенных узлах, задайте `nodeSelector` и `tolerations` для `ClusterALBInstance`.

Если модуль `alb` работает за внешним L7-балансировщиком (например, Cloudflare или Qrator), включите `useProxyProtocol`, чтобы получать реальные адреса клиентов. Дополнительно ограничьте с помощью `spec.originalIPDetection` список подсетей, из которых разрешено доверять заголовкам с адресом клиента.

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: main
spec:
  gatewayName: public-gw
  inlet:
    type: HostPort
    hostPort:
      httpPort: 80
      httpsPort: 443
  useProxyProtocol: true
  originalIPDetection:
    setRealIPFrom:
      - 10.0.0.0/16

```



Proxy Protocol и HTTP/3 нельзя включать одновременно: если задать оба параметра, контроллер модуля `alb` отклонит конфигурацию как конфликтующую.

Смена инлета с сохранением текущего Gateway

Чтобы сменить инлет, используемый для уже созданного объекта Gateway, выполните следующие действия:

1. Создайте новый объект `ClusterALBInstance` или объект `ALBInstance` с другим именем, но с тем же значением `spec.gatewayName`, используя нужный тип инлета.
2. Проверьте, что новый путь приёма трафика работает корректно.
3. Удалите неактуальный объект `ClusterALBInstance` или объект `ALBInstance`.

Так как `gatewayName` не меняется, объект Gateway остаётся прежним. В большинстве случаев объект `ListenerSet` и маршруты при этом можно не пересоздавать.

Открытие дополнительного TCP/UDP-порта на общекластерном Gateway

Если кроме стандартных HTTP/HTTPS-слушателей на шлюзе нужен отдельный TCP/UDP-порт, добавьте в соответствующий `ClusterALBInstance` поле `spec.inlet.additionalPorts`, например:

```

spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
    additionalPorts:
      - port: 9000
        protocol: TCP

```

Контроллер добавит в управляемый объект Gateway слушатель TCP/UDP-трафика с именем секции (`sectionName`), например, `tcp-port-9000`. Чтобы привязать к нему `TCPRoute`, укажите в маршруте имя Gateway и соответствующее значение `sectionName`:

```

apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: app-tcp
  namespace: prod
spec:
  parentRefs:
    - name: public-gw
      namespace: d8-alb
      sectionName: tcp-port-9000
      port: 9000
  rules:
    - backendRefs:
        - name: tcp-svc
          port: 9000

```

i Если объект TCPRoute или UDPRoute создаётся в неймспейсе, отличном от неймспейса целевого Gateway, дополнительно создайте в неймспейсе Gateway объект ReferenceGrant, разрешающий ссылки из неймспейса маршрута.

Примеры UDPRoute и шаги публикации приложений — в разделе «Работа с объектами GRPCRoute, TLSRoute, TCPRoute и UDPRoute».

Разделение публичной и административной зон

Для публичного и административного трафика можно использовать отдельные объекты Gateway. Это позволяет задать для каждой зоны отдельную точку входа и собственные правила доступа. Такой подход аналогичен использованию отдельного Ingress NGINX Controller и IngressClass для административной зоны.

Создайте отдельный объект Gateway для каждой зоны и ограничьте приём административного трафика с помощью `spec.acceptRequestsFrom`. Решение о допуске соединения принимается по реальному адресу подключения, а не по заголовкам запроса.

Пример создания двух объектов ClusterALBInstance — для публичной и административной зон:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
---
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: admin
spec:
  gatewayName: admin-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16

```

Далее для каждого шлюза создайте отдельные объекты ListenerSet и маршруты. Примеры публикации приложений приведены в разделе «Публикация приложения с ListenerSet и HTTPRoute».

Публикация в неймспейсе (ALBInstance)

Пример минимальной рабочей конфигурации ALBInstance для публикации шлюза в неймспейсе:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```

После перехода ALBInstance в состояние `Ready` создайте в том же неймспейсе объекты `ListenerSet` и `HTTPRoute`. Порядок их настройки и пример публикации приложения приведены в разделе «Публикация приложения с `ListenerSet` и `HTTPRoute`».

Выпуск TLS-сертификатов с cert-manager

Модуль `alb` использует `cert-manager` для автоматического выпуска TLS-сертификатов. Слушатели `d8-http` и `d8-https` обслуживают запросы HTTP-01 (HTTP-01 challenge) при выпуске сертификата. Чтобы выпустить сертификат для приложения, создайте объект `Certificate`, который сохранит сертификат в `Secret`, и укажите этот `Secret` в `certificateRefs` соответствующего `ListenerSet`.

Пример сертификата, который создаёт `Secret app-tls` для `ListenerSet`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: app-tls
  namespace: prod
spec:
  secretName: app-tls
  issuerRef:
    name: letsencrypt
    kind: ClusterIssuer
  dnsNames:
    - app.example.com
```

Сертификаты выпускает модуль `cert-manager`. Если модуль `alb` используется вместе с `ingress-nginx`, используйте отдельный `ClusterIssuer` для каждого типа ALB, как описано в разделе «Совместно с `ingress-nginx`».

Совместное использование с другими модулями и сторонними решениями

ALB средствами Kubernetes Gateway API в кластере DKP можно использовать совместно с ALB средствами Ingress NGINX Controller, а также с ALB на основе сторонних решений Gateway API. Пошаговый переход — в разделе «Миграция с `ingress-nginx` на `alb`».

Совместно с ingress-nginx

ALB средствами Kubernetes Gateway API может использоваться в кластере совместно с «ALB средствами Ingress NGINX Controller». В таком случае для каждого из типов ALB рекомендуется использовать отдельный объект `ClusterIssuer`, чтобы раздельно управлять настройками и жизненными циклами сертификатов. Один и тот же внешний `hostname` не должен одновременно обслуживаться обоими ALB без разделения на уровне DNS или внешнего балансировщика.



Для шлюза DKP по умолчанию объект `ClusterIssuer` создаётся автоматически. Этот же объект `ClusterIssuer` используется для выпуска сертификатов системных доменов.

Совместно со сторонними реализациями Gateway API

Использование сторонних решений Gateway API допускается при условии, что в кластере используются следующие, совместимые с контроллером модуля `alb`, версии API для объектов Gateway API:

- BackendTLPolicy: v1;
- GatewayClass: v1;
- Gateway: v1;
- ListenerSet: v1;
- GRPCRoute: v1;
- HTTPRoute: v1;
- ReferenceGrant: v1beta1;
- TCPRoute: v1alpha2/v1;
- UDPRoute: v1alpha2/v1;
- TLSRoute: v1.

Контроллер модуля `alb` в процессе запуска проверяет текущие хранимые версии объектов Gateway API. В случае обнаружения расхождения между установленными и требуемыми версиями контроллер прекращает работу. Если же в кластере полностью отсутствует тот или иной тип объекта Gateway API, нужная версия будет создана контроллером автоматически и он продолжит работу.

Для ручной проверки совместимости версий установленных в кластере объектов Gateway API с требуемыми версиями используйте команду:

```
declare -A want=(
  [gatewayclasses.gateway.networking.k8s.io]=v1
  [gateways.gateway.networking.k8s.io]=v1
  [grpcroutes.gateway.networking.k8s.io]=v1
  [httproutes.gateway.networking.k8s.io]=v1
  [listenersets.gateway.networking.k8s.io]=v1
  [referencegrants.gateway.networking.k8s.io]=v1beta1
  [tcproutes.gateway.networking.k8s.io]="v1|v1alpha2"
  [udproutes.gateway.networking.k8s.io]="v1|v1alpha2"
  [tlsroutes.gateway.networking.k8s.io]=v1
  [backendtlspolicies.gateway.networking.k8s.io]=v1
)

for crd in "${!want[@]}"; do
  got="$(
    d8 k get crd "$crd" \
      -o jsonpath='{.spec.versions[?(@.storage==true)].name}' \
      2>/dev/null || true
  )"
  if [[ -n "$got" && "$got" =~ ^(${want[$crd]})$ ]]; then
    echo "$crd OK storage=$got"
  else
    echo "$crd FAILED cluster=${got:-MISSING} expected=${want[$crd]}"
  fi
done | sort
```

В остальном модуль конфигурирует и управляет только объектами Gateway определённого GatewayClass, что минимизирует риск возникновения конфликтов при использовании сторонних решений Gateway API.

Включение HTTP/3

По умолчанию шлюз принимает HTTP/1.1 и HTTP/2. Чтобы включить HTTP/3, задайте `spec.enableHTTP3: true` в `ClusterALBInstance` или `ALBInstance`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
spec:
  gatewayName: public-gw
  enableHTTP3: true
  inlet:
    type: LoadBalancer
    loadBalancer: {}
```



Параметры `enableHTTP3` и `useProxyProtocol` нельзя включать одновременно: если задать оба параметра, контроллер модуля `alb` отклонит конфигурацию как конфликтующую.

Учитывайте следующее:

- HTTP/3 использует QUIC поверх UDP. Убедитесь, что балансировщик и сетевые правила пропускают UDP на порт HTTPS (обычно 443), а не только TCP.
- Для инлета `LoadBalancer` проверьте, что облачный или MetalLB-балансировщик принимает UDP на нужном порту.
- Клиенты должны поддерживать HTTP/3. Проверка с рабочей станции:

```
curl -vk --http3 https://app.example.com/
```

Если `curl` собран без HTTP/3, используйте клиент с поддержкой QUIC или проверьте доступность UDP-порта и журналы Envoy Proxy.

Использование GeoIP и GeoLite2

Модуль `alb` поддерживает добавление полей GeoIP в заголовки HTTP-запросов на основе данных баз MaxMind GeoIP/GeoLite2.

На данный момент возможно подключение следующих редакций баз:

- GeoIP2-Anonymous-IP;
- GeoIP2-City;
- GeoIP2-ISP;
- GeoIP2-ASN;
- GeoLite2-ASN;
- GeoLite2-City.



Текущая интеграция GeoIP поддерживает одновременное использование до 4 баз.

Выберите способ получения баз GeoIP:

- Скачивание баз с MaxMind
- Локальное зеркало

Для подключения GeoIP и скачивания баз непосредственно с серверов MaxMind необходимо предварительно создать Secret, содержащий лицензионный ключ, например:

```
d8 k -n prod create secret generic geoip-license \
  --from-literal=licenseKey='<MAXMIND_LICENSE_KEY>'
```



При настройке GeoIP для `ClusterALBInstance` Secret может быть размещён в любом неймспейсе, но рекомендуется разместить его в `d8-alb`.

Для объектов `ALBInstance` Secret должен располагаться строго в том же неймспейсе, что и объект `ALBInstance`.

После создания Secret укажите его в параметре `spec.geoIP.licenseKeySecretRef` объекта `ClusterALBInstance` или `ALBInstance`, например:


```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    licenseKeySecretRef:
      name: geoip-license

```

Для подключения GeoIP и скачивания баз с локального зеркала укажите его адрес в параметре `spec.geoIP.maxmindMirror.url`, например:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    maxmindMirror:
      url: "https://local.geoip:8443"

```

В качестве URL допускается указание адреса локального кеширующего сервера GeoIP в другом неймспейсе, например:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    maxmindMirror:
      url: "http://geoproxy-cluster.d8-alb.svc:8080/download"

```

Использование заголовков GeoIP

В результате настройки GeoIP в неймспейсе, где работают прокси ClusterALBInstance или ALBInstance, будет запущен сервер кеширования и обновления баз GeoIP. Поды Envoy Proxy затем поочерёдно перезапускаются, чтобы скачивать базы с локального сервера GeoIP.

Для добавления полей GeoIP в заголовки HTTP-запросов укажите имена HTTP-заголовков в параметре `spec.geoIP.headers`, например:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: main
  namespace: prod
spec:
  envoyLogLevel: Warning
  gatewayName: custom-gateway
  geoIP:
    headers:
      city: geoip_city
      country: geoip_country
    licenseKeySecretRef:
      name: geoip-license
    maxmindEditionIDs:
      - GeoLite2-City

```

Обновление баз GeoIP осуществляется раз в сутки: сначала на кеширующем сервере, а затем в каждом отдельном поде Envoy Proxy, который обращается к этому серверу.

Параметр `storageClass` модуля `alb` задаёт PVC для компонентов GeoIP.

Настройка трассировки OpenTelemetry

Модуль `alb` поддерживает экспорт трассировок OpenTelemetry из Envoy Proxy.

Для включения экспорта укажите адрес OpenTelemetry Collector в параметре `spec.openTelemetry.tracing.url` — в формате `http://`, `https://` или `grpc://`, с указанием хоста, порта и, при необходимости, пути (например, для OTLP/HTTP).

Долю запросов, которые Envoy Proxy случайно выбирает для трассировки, задаёт параметр `randomSamplingPercentage` (по умолчанию — 25%).

При необходимости настройте подключение с использованием TLS.

При использовании TLS рекомендуется явно задать параметр `sni`, если OpenTelemetry Collector находится за прокси или балансировщиком, который выбирает upstream на основе Server Name Indication.

Настройка TLS для OpenTelemetry

Чтобы передавать данные трассировки OpenTelemetry по TLS, создайте Secret с CA-сертификатом и укажите его в параметре `spec.openTelemetry.tracing.tls.caSecretName`.

- Для ClusterALBInstance или шлюза DKP по умолчанию разместите Secret в неймспейсе `d8-alb`.
- Для ALBInstance разместите Secret в том же неймспейсе, что и объект ALBInstance.

CA-сертификат должен быть сохранён в ключе `casert`. Дополнительные Subject Alternative Names для проверки сертификата OpenTelemetry Collector задаются параметром `subjectAltNames`, а параметр `insecureSkipVerify` отключает эту проверку.

```

apiVersion: v1
kind: Secret
metadata:
  name: otel-tracing-ca
  namespace: d8-alb
type: Opaque
stringData:
  cacert: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
---
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: proxy-gw
spec:
  gatewayName: proxy-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
  openTelemetry:
    tracing:
      url: https://otel-collector.monitoring.svc.cluster.local:4318/v1/traces
      tls:
        sni: otel-collector.monitoring.svc.cluster.local
        caSecretName: otel-tracing-ca

```

Диагностика и проверка

В этом разделе — проверка готовности шлюза и типичные вопросы при первой настройке. Дополнительные сценарии — в «FAQ модуля alb».

- После создания ClusterALBInstance или ALBInstance дождитесь состояния `Ready` и возьмите имя и неймспейс Gateway из `status`. Описание полей — в ClusterALBInstance status и ALBInstance status:

```

d8 k get clusteralbinstance
d8 k -n <NAMESPACE> get albinstance
d8 k -n d8-alb get gateway

```

- При инлете LoadBalancer возьмите адрес у сервиса балансировщика (обычно в неймспейсе `d8-alb`) и укажите его в DNS для hostname из ListenerSet:

```

d8 k -n d8-alb get svc

```

- Проверьте, не используется ли один и тот же hostname для двух ALB. Модули `alb` и `ingress-nginx` могут работать в одном кластере, но один и тот же внешний hostname не должен одновременно обслуживаться двумя ALB без явного разделения на уровне DNS или внешнего балансировщика.
- Проверьте, нет ли конфликта в объектах ListenerSet. Если два ListenerSet с одинаковыми обработчиками ссылаются на один Gateway, контроллер отклонит конфликтующую конфигурацию. Измените hostname, порт или протокол либо удалите дублирующий ListenerSet.
- Для диагностики, можно посмотреть конфигурацию, переданную контроллером и конфигуратором прокси в Envoy Proxy, обслуживающий объект Gateway.

- Выберите под Envoy Proxy для нужного объекта Gateway:

```

d8 k -n d8-alb get pods -l alb.deckhouse.io/gateway=shared-gateway

```

- Получите конфигурацию пода с помощью команды (вместо `<ENVOY_PROXY_POD_NAME>`

используйте имя пода Envoy Proxy, полученное на предыдущем шаге):

```
d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump
```

Если нужен только отдельный раздел конфигурации, явно укажите его:

```
d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
resource=dynamic_listeners
d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
resource=dynamic_route_configs
d8 k -n d8-alb exec -it <ENVOY_PROXY_POD_NAME> -- pilot-agent request GET /config_dump?
resource=dynamic_active_clusters
```

Так можно проверить, появились ли ожидаемые обработчики трафика, виртуальные хосты и upstream-кластеры после изменения объекта ListenerSet или объекта Route.

Дополнительные ресурсы

- Миграция с ingress-nginx на alb
- Публикация приложений средствами Kubernetes Gateway API
- Балансировка входящего трафика
- Документация модуля alb

Миграция с ingress-nginx на alb

В руководстве описана миграция с модуля `ingress-nginx` на модуль `alb`. В рамках этой миграции публикация приложений переходит с Ingress API на Gateway API.

Руководство охватывает различия моделей, подготовку инфраструктуры модуля `alb` с ClusterALBInstance или ALBInstance и управляемым Gateway, перевод приложений на Gateway API, миграцию системных интерфейсов Deckhouse Kubernetes Platform (DKP), переключение трафика на модуль `alb` и откат при необходимости.

Порядок работ:

1. Подготовка инфраструктуры модуля alb.
2. Миграция интерфейсов DKP — только если системные интерфейсы сейчас на Ingress и их нужно перевести на Gateway API.
3. Миграция публикации приложений.
4. Переключение трафика на модуль alb.
5. Очистка.

Причины перехода на Gateway API

Основные причины перехода с Ingress API на Gateway API:

- Активное сопровождение upstream-проекта Ingress NGINX, используемого в DKP, прекращено. Новые функции, исправления и интеграции в нём не ожидаются. Для новых сценариев публикации приложений используйте Gateway API.
- В отличие от Ingress API, Gateway API описывает маршруты отдельными ресурсами по протоколам, явно настраивает точки приёма трафика, контролирует подключение маршрутов и доступ между неймспейсами. Сложные конфигурации можно задавать ресурсами API, а не только аннотациями конкретного контроллера.
- Gateway API разделяет ответственность между ролями. Администраторы кластера и сети управляют инфраструктурой обработки трафика и объектами Gateway через ClusterALBInstance или ALBInstance. Администраторы неймспейса настраивают приём трафика через ListenerSet (hostname, TLS, порты). Разработчики приложения задают маршрутизацию с помощью HTTPRoute и других ресурсов маршрутов. Такое разделение позволяет делегировать настройки, проверять конфигурацию и выполнять поэтапную миграцию.

Сравнение моделей

В этом разделе сравнивается, как описывается инфраструктура обработки трафика, как DKP создаёт её и как правила маршрутизации приложений преобразуются в конфигурацию прокси-сервера.

Ingress API и ingress-nginx

Ниже представлена схема ресурсов и прохождения трафика через модуль `ingress-nginx`.

Обработка HTTP- и HTTPS-трафика через модуль `ingress-nginx` настраивается следующим образом:

1. Администратор кластера создаёт кластерный объект `IngressNginxController`.
2. В `IngressNginxController` указывается имя используемого объекта `IngressClass`. Если имя не задано, используется `nginx`.
3. DKP обрабатывает объект и создаёт необходимую инфраструктуру, включая `IngressClass`. Несколько объектов `IngressNginxController` могут использовать один `IngressClass`.
4. По умолчанию ресурсы DKP публикуются через `IngressClass` с именем `nginx`. Другой класс можно выбрать в глобальной конфигурации DKP.
5. Администраторы сети или разработчики приложения создают объекты `Ingress`, которые явно или неявно выбирают требуемый `IngressClass`.
6. Итоговая конфигурация `nginx` объединяет параметры инфраструктуры из `IngressNginxController` с объектами `Ingress`, выбранными по `IngressClass`.

Gateway API и alb

Ниже представлена схема ресурсов и прохождения трафика через модуль `alb`.

Обработка трафика HTTP, HTTPS, gRPC, TLS, TCP и UDP через модуль `alb` настраивается следующим образом:

1. Администратор кластера создаёт кластерный объект `ClusterALBInstance`.
2. В `ClusterALBInstance` задаются параметры инфраструктуры и обязательный параметр `gatewayName`, который определяет управляемый объект `Gateway`.
3. Контроллер модуля `alb` обрабатывает объект и создаёт управляемый `Gateway` и необходимую инфраструктуру обработки трафика. Несколько объектов `ClusterALBInstance` могут использовать одинаковое значение `gatewayName` и, следовательно, один объект `Gateway`.
4. Если настроен объект `Gateway` DKP по умолчанию, модули DKP создают для него объекты `ListenerSet`, `HTTPRoute` и другие ресурсы `Gateway API`.
5. Администраторы сети или разработчики приложения создают ресурсы `Gateway API` и подключают их к управляемому объекту `Gateway`.
6. Итоговая конфигурация `Envoy Proxy` объединяет параметры инфраструктуры из `ClusterALBInstance` с конфигурацией, заданной ресурсами `Gateway API`, подключёнными к `Gateway`.

Ключевые архитектурные различия

В таблице приведены следующие архитектурные различия:

Параметр	Ingress API и ingress-nginx	Gateway API и alb
Связи между ресурсами	Инфраструктура и правила маршрутизации косвенно связаны через IngressClass	Ресурсы образуют явный граф с помощью parentRefs и других типизированных ссылок. Корневым объектом является Gateway
Разделение ответственности	IngressNginxController определяет инфраструктуру, а Ingress объединяет маршрутизацию приложения с настройками, зависящими от контроллера	Администраторы кластера и сети управляют экземплярами и объектами Gateway, администраторы неймспейса — ListenerSet, разработчики приложения — HTTPRoute и другие ресурсы маршрутов
Модель конфигурации	Ingress объединяет основную конфигурацию маршрутизации HTTP- и HTTPS-трафика в одном объекте и использует аннотации, зависящие от контроллера	Слушатели, маршруты, бэкенды и политики представлены отдельными ресурсами, формирующими общую конфигурацию
Общая конфигурация и точки входа	Несколько объектов IngressNginxController, использующих один IngressClass, применяют общий набор правил Ingress, но предоставляют отдельные точки входа	Несколько объектов ClusterALBInstance с одинаковым значением gatewayName предоставляют отдельные точки входа на основе общей конфигурации Gateway
Ссылки между неймспейсами	Ingress, объект Service, указанный в качестве бэкенда, и объект Secret с TLS-сертификатом, как правило, находятся в одном неймспейсе	Доступ к ресурсам в других неймспейсах явно разрешается с помощью ReferenceGrant
Поддержка протоколов	Ingress предназначен преимущественно для HTTP- и HTTPS-трафика	Для HTTP, gRPC, TCP, TLS и UDP предусмотрены отдельные типы маршрутов
Расширение функциональности	Дополнительные параметры обычно задаются аннотациями, зависящими от реализации контроллера	Больше параметров задаётся с помощью структурированных и проверяемых ресурсов API и политик
Жизненный цикл и владение	Возможности делегирования конфигурации инфраструктуры и маршрутизации, а также управления их связями ограничены	Инфраструктура Gateway может оставаться неизменной, пока разработчики приложения независимо создают, изменяют и удаляют маршруты
Подключение маршрутов	Выбор IngressClass задаёт общую связь между объектом Ingress и контроллерами	Объекты Gateway и их слушатели явно определяют, какие маршруты могут к ним подключаться



Несколько объектов ClusterALBInstance могут ссылаться на один Gateway, но параметры уровня Gateway, например, additionalPorts, могут конфликтовать. Используйте совместимые параметры Gateway во всех экземплярах, связанных с одним объектом Gateway.

В соответствии с моделью разрешения конфликтов Gateway API модуль alb формирует итоговую конфигурацию по объекту ClusterALBInstance с наиболее ранней датой создания. Конфликтующие параметры более новых объектов игнорируются, а сведения о конфликте отображаются в их статусе.

Шаг 1. Подготовка инфраструктуры модуля alb

На этом шаге выберите тип экземпляра модуля `alb` (`ClusterALBInstance` или `ALBInstance`), настройте инлет и подготовьте TLS-сертификаты для точки входа Gateway API.

Выбор ALBInstance или ClusterALBInstance

При выборе типа экземпляра учитывайте следующее:

- Используйте `ClusterALBInstance` для общего или платформенного объекта Gateway, публикации системных интерфейсов DKP, а также если требуется инлет `HostPort`.
- Для публикации системных интерфейсов DKP выполните инструкции из раздела «Публикация служебных доменов».
- Используйте `ALBInstance` для объекта Gateway, выделенного приложению или команде и управляемого в пределах соответствующего неймспейса. `ALBInstance` поддерживает инлеты `LoadBalancer` и `ClusterIP`. Для миграции с `ingress-nginx` актуален инлет `LoadBalancer`.
- Подробное сравнение приведено в разделе «ClusterALBInstance и ALBInstance».

Настройка инлета

В `IngressNginxController` тип инлета одновременно определяет способ приёма трафика и дополнительные функции, например, Proxy Protocol и сквозную передачу TLS-трафика. В модуле `alb` эти параметры настраиваются отдельно:

- `ClusterALBInstance` поддерживает инлеты `LoadBalancer` и `HostPort`.
- `ALBInstance` поддерживает инлеты `LoadBalancer` и `ClusterIP`.
- Proxy Protocol включается параметром `spec.useProxyProtocol`. Его можно включать и отключать в существующем экземпляре без перезапуска подов `Envoy Proxy` или пересоздания экземпляра.
- Сквозная передача TLS-трафика настраивается с помощью TLS-слушателя и объекта `TLSRoute`, а не отдельного типа инлета.

При выборе инлета для модуля `alb` используйте следующее соответствие:

Инлет <code>IngressNginxController</code>	Конфигурация модуля <code>alb</code>	Особенности миграции
<code>LoadBalancer</code>	<code>ClusterALBInstance</code> или <code>ALBInstance</code> с <code>spec.inlet.type: LoadBalancer</code>	Контроллер создаёт объект <code>Service</code> с типом <code>LoadBalancer</code>
<code>LoadBalancerWithProxyProtocol</code>	Инлет <code>LoadBalancer</code> с <code>spec.useProxyProtocol: true</code>	Настройте внешний балансировщик для передачи Proxy Protocol. Proxy Protocol и HTTP/3 нельзя включить одновременно. Контроллер модуля <code>alb</code> отклонит такую конфигурацию как конфликтующую
<code>LoadBalancerWithSSLPassthrough</code>	Инлет <code>LoadBalancer</code> с TLS-слушателем и объектом <code>TLSRoute</code>	Сквозная передача TLS-трафика относится к конфигурации маршрутизации Gateway API и не является отдельным вариантом инлета
<code>HostPort</code>	<code>ClusterALBInstance</code> с <code>spec.inlet.type: HostPort</code>	Инлет <code>HostPort</code> не поддерживается для <code>ALBInstance</code>
<code>HostPortWithProxyProtocol</code>	<code>ClusterALBInstance</code> с инлетом <code>HostPort</code> и <code>spec.useProxyProtocol: true</code>	Proxy Protocol и HTTP/3 нельзя включить одновременно
<code>HostPortWithSSLPassthrough</code>	<code>ClusterALBInstance</code> с инлетом <code>HostPort</code> , TLS-слушателем и объектом <code>TLSRoute</code>	Сквозная передача TLS-трафика настраивается независимо от инлета

Инлет IngressNginxController	Конфигурация модуля alb	Особенности миграции
HostWithFailover	Прямой аналог отсутствует	Используйте ClusterALBInstance с инлетом LoadBalancer и балансировщиком MetalLB. Выполните настройку по «Пример для bare metal с балансировщиком MetalLB» и проверьте отказоустойчивость балансировщика до переключения пользовательского трафика



Во время миграции модуль ingress-nginx с инлетом HostPort или HostWithFailover и модуль alb с инлетом HostPort не могут использовать одинаковые порты хоста на одних и тех же узлах. Выберите разные группы узлов или другой набор портов хоста для одного из контроллеров.

Связанные параметры инлета переносятся следующим образом:

IngressNginxController	ClusterALBInstance или ALBInstance
spec.loadBalancer.annotations	spec.inlet.loadBalancer.serviceAnnotations
spec.loadBalancer.loadBalancerClass	spec.inlet.loadBalancer.loadBalancerClass
spec.loadBalancer.httpPort, httpsPort	spec.inlet.loadBalancer.httpPort, httpsPort
spec.loadBalancer.sourceRanges	spec.inlet.loadBalancer.loadBalancerSourceRanges
spec.hostPort.httpPort, httpsPort	spec.inlet.hostPort.httpPort, httpsPort
spec.acceptRequestsFrom	spec.acceptRequestsFrom
spec.*.behindL7Proxy, realIPHeader, acceptClientIPHeadersFrom	spec.originalIPDetection.realIPHeader, setRealIPFrom

При переносе параметров учитывайте следующее:

- Перенесите только аннотации объекта Service, поддерживаемые целевой реализацией балансировщика.
- Задайте параметр spec.inlet.loadBalancer.loadBalancerClass при создании объекта. После создания его изменить нельзя.
- Оставьте порты HTTP 80 и HTTPS 443 по умолчанию либо задайте для порта значение 0 в модуле alb, чтобы отключить соответствующий слушатель по умолчанию.
- Настройте параметры HostPort только для ClusterALBInstance. Укажите хотя бы один порт.
- Перенесите необходимые ограничения доступа по исходным CIDR-диапазонам в spec.acceptRequestsFrom.
- Явно укажите CIDR доверенных прокси-серверов. Не разрешайте обработку заголовков с адресом клиента от произвольных источников.
- Проверьте работу loadBalancerSourceRanges с целевой реализацией балансировщика. Значение передаётся в объект Service типа LoadBalancer. Облачный провайдер может не поддерживать или игнорировать этот параметр.

Тип инлета нельзя изменить после создания ни для ClusterALBInstance, ни для ALBInstance. Чтобы сменить инлет, создайте новый экземпляр с тем же gatewayName, проверьте трафик, переключите на него нагрузку и удалите старый экземпляр.

TLS и сертификаты



Если `ingress-nginx` и `alb` используются одновременно, а сертификаты выпускаются объектами `Issuer` или `ClusterIssuer` с механизмом проверки `HTTP-01`, используйте отдельные ресурсы `Certificate` и отдельные объекты `Secret` для путей `Ingress API` и `Gateway API`. Иначе возможны конфликты при выпуске или продлении сертификата.

Для объекта `Gateway DKP` по умолчанию `DKP` создаёт отдельный `ClusterIssuer` с механизмом проверки `HTTP-01 Let's Encrypt`. Отдельные `Certificate` и `Secret` нужны только для объектов `Issuer` и `ClusterIssuer` с `HTTP-01` и не требуются для объектов, настроенных исключительно на `DNS-01`.

Инструкции по настройке `Issuer` или `ClusterIssuer` с механизмом проверки `HTTP-01` через `Gateway API` приведены в разделе «Добавление собственного `HTTP-01 ClusterIssuer` или `Issuer` для `ALB`».

Чтобы завершить подготовку инфраструктуры, выполните следующие действия:

1. Создайте `ClusterALBInstance` или `ALBInstance` с выбранным инлетом и параметрами из таблиц выше.
2. Дождитесь готовности экземпляра и управляемого объекта `Gateway`.
3. Подготовьте отдельные ресурсы `Certificate` и объекты `Secret` для пути `Gateway API`, если одновременно используются `Issuer` или `ClusterIssuer` с механизмом проверки `HTTP-01`.

Шаг 2. Миграция интерфейсов DKP

Если системные интерфейсы `DKP` публикуются через `Ingress` и их нужно перевести на `Gateway API`, выполните инструкции из раздела «Публикация служебных доменов».



Не все модули `DKP` пока публикуют служебные `HTTPRoute` через `Gateway API`. Команда `jq` из этого раздела покажет, какие модули уже делают это в вашем кластере, — это фактический список опубликованных маршрутов, а не полный перечень возможностей платформы.

Если миграция интерфейсов `DKP` не требуется, перейдите к шагу 3.

Шаг 3. Миграция публикации приложений

Список ресурсов `Gateway API`, поддерживаемых модулем `alb`, и сведения о владении управляемыми объектами `Gateway` приведены в разделе «Поддерживаемые ресурсы `Gateway API`».

Конвертация Ingress в Gateway API

Чтобы преобразовать ресурсы `Ingress` в ресурсы `Gateway API`, выполните следующие действия:

1. Убедитесь, что целевой `ALBInstance` или `ClusterALBInstance` создан на шаге 1 и готов.
2. Получите неймспейс и имя управляемого объекта `Gateway` из статуса экземпляра.
3. Преобразуйте каждый объект `Ingress` и связанную с ним конфигурацию в соответствующие ресурсы `ListenerSet`, маршруты и политики для этого `Gateway`. Встроенный инструмент конвертации даёт черновик, однако не для каждой функции `ingress-nginx` есть прямой аналог в `Gateway API`.
4. Перед применением проверьте сгенерированные манифесты и при необходимости отредактируйте их.
5. Примените ресурсы:

```
d8 k apply -f gateway-api.yaml
```

6. Проверьте статусы `Gateway`, `ListenerSet` и маршрутов до переключения трафика.

Использование встроенного инструмента `ingress2gateway`

Контроллер `Gateway` предоставляет `HTTP`-эндпоинт, который принимает один объект, объект `Kubernetes List` или `YAML` с несколькими документами. Конвертер обрабатывает следующие входные ресурсы:

- `Ingress` — ресурсы, выбранные для преобразования.
- `Service` — ресурсы, используемые для определения именованных портов и метаданных объектов `Service`.
- `DexAuthenticator` — используется для определения `spec.applicationDomain`, если

конфигурация внешней аутентификации `ingress-nginx` содержит переменные `nginx`, например, `$host`.

В качестве входных данных передайте все связанные ресурсы, чтобы конвертер мог сохранить поддерживаемые параметры их конфигурации. По умолчанию эндпоинт отключён и доступен только внутри подов контроллера Gateway.

1. Временно включите параметр `migrations.ingress2Gateway.enabled` в конфигурации модуля `alb` и дождитесь перезапуска контроллера Gateway:

```
d8 k patch moduleconfig alb --type merge \
  --patch '{"spec":{"settings":{"migrations":{"ingress2Gateway":
{"enabled":true}}}}}'
d8 k -n d8-alb rollout status deployment/gateway-controller
```

2. Перенаправьте порт эндпоинта из пода контроллера Gateway:

```
d8 k -n d8-alb port-forward deployment/gateway-controller 8082:8082
```

3. В другом терминале экспортируйте все распознаваемые типы ресурсов и передайте полученный Kubernetes List непосредственно конвертеру:

```
converter_url='http://127.0.0.1:8082/ingress2gateway'
converter_url="${converter_url}?gateway=<GATEWAY_NAMESPACE>/<GATEWAY_NAME>"
converter_url="${converter_url}&scope=<SCOPE>&ingress-class=<INGRESS_CLASS>"

d8 k get ingress,service,dexauthenticator \
  --all-namespaces --output yaml | \
  curl --fail-with-body --silent --show-error --request POST \
    --header 'Content-Type: application/yaml' \
    --data-binary @- \
    --output gateway-api.yaml \
    "$converter_url"
```

В URL имена параметров запроса пишутся строчными буквами: `gateway`, `scope`, `ingress-class`. В примере их значения заданы плейсхолдерами:

- `<GATEWAY_NAMESPACE>` — неймспейс управляемого объекта Gateway из статуса экземпляра на шаге 1;
 - `<GATEWAY_NAME>` — имя управляемого объекта Gateway из того же статуса.
- Пара `<GATEWAY_NAMESPACE>/<GATEWAY_NAME>` — значение параметра `gateway`. Используйте значение по умолчанию `d8-alb/public-gw` только если оно совпадает с вашим Gateway.
- `<SCOPE>` — `cluster` для кластерной модели или `namespaced` для Gateway в неймспейсе приложения (как у ALBInstance). Если параметр не указать, используется `cluster`;
 - `<INGRESS_CLASS>` — IngressClass исходных ресурсов Ingress. Если параметр не указать, используется `nginx`.



Размер тела запроса ограничен 8 МБ. В крупных кластерах экспортируйте только переносимые ресурсы и связанные с ними объекты из нужных неймспейсов.

4. Перед применением проверьте файл `gateway-api.yaml` и диагностические сообщения конвертера, добавленные в виде комментариев YAML, затем примените ресурсы.
5. После завершения преобразования отключите эндпоинт:

```
d8 k patch moduleconfig alb --type merge \
  --patch '{"spec":{"settings":{"migrations":{"ingress2Gateway":
{"enabled":false}}}}}'
```

Расширение Gateway API с помощью аннотаций

Спецификация Gateway API не охватывает все зависящие от реализации функции управления трафиком, необходимые для работы DKP. Поэтому модуль `alb` использует аннотации HTTPRoute для параметров, которые пока нельзя задать стандартными полями Gateway API.

По мере развития Gateway API модуль `alb` будет постепенно заменять аннотации соответствующими стандартными ресурсами и полями.

При миграции по возможности используйте стандартные поля Gateway API, а аннотации `ingress-nginx` заменяйте только поддерживаемыми аннотациями `alb` — в том числе после применения `gateway-api.yaml`. Актуальный список приведён в разделе «Поддерживаемые аннотации HTTPRoute».

Утилита `ingress2gateway` формирует черновик ресурсов Gateway API и не переносит все аннотации `ingress-nginx` автоматически. После преобразования сверьте Ingress с таблицей ниже и перенесите нужные параметры вручную.

Аннотация Ingress (<code>ingress-nginx</code>)	Эквивалент в модуле <code>alb</code>
<code>nginx.ingress.kubernetes.io/whitelist-source-range</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/whitelist-source-range</code>
<code>nginx.ingress.kubernetes.io/service-upstream</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/service-upstream</code>
<code>nginx.ingress.kubernetes.io/upstream-vhost</code>	Фильтр URLRewrite с <code>hostname</code> в HTTPRoute (публикация с Istio-сайдкаром)
<code>nginx.ingress.kubernetes.io/auth-url / auth-signin</code>	Аннотации HTTPRoute <code>alb.network.deckhouse.io/auth-url</code> и <code>alb.network.deckhouse.io/auth-signin</code>
<code>nginx.ingress.kubernetes.io/auth-type: basic</code> и <code>Secret</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/basic-auth-secret</code>
<code>nginx.ingress.kubernetes.io/limit-rps</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/limit-rps</code>
<code>nginx.ingress.kubernetes.io/proxy-body-size</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/buffer-max-request-bytes</code> (значение в байтах)
<code>nginx.ingress.kubernetes.io/proxy-buffer-size</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/proxy-buffer-size</code>
<code>nginx.ingress.kubernetes.io/proxy-read-timeout / proxy-send-timeout</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/idle-timeout</code> (тайм-аут неактивности, не полная длительность запроса)
<code>nginx.ingress.kubernetes.io/affinity / cookie</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/session-affinity</code>
<code>nginx.ingress.kubernetes.io/rewrite-target</code>	Аннотация HTTPRoute <code>alb.network.deckhouse.io/rewrite-target</code> или стандартные фильтры Gateway API
<code>nginx.ingress.kubernetes.io/configuration-snippet</code> и прочие сниппеты <code>nginx</code>	Прямого аналога нет; пересмотрите конфигурацию под возможности Gateway API и аннотации <code>alb</code>

Если аннотации Ingress нет в таблице и нет поля Gateway API, поведение после миграции может отличаться или пропасть. Проверьте приложение на пути модуля `alb` до переключения пользовательского трафика.

Шаг 4. Переключение трафика на модуль `alb`

Используйте `ingress-nginx` и `alb` одновременно, пока обработка трафика через модуль `alb` не будет проверена и не завершится период, предусмотренный для отката.

Переносите отдельные домены или неймспейсы, если для них можно использовать разные DNS-записи. В остальных случаях переключайте общую внешнюю точку входа, изменяя DNS-записи

или пул бэкендов внешнего балансировщика.

На этом шаге:

- протестируйте модуль `alb` без смены DNS через `migrationGateway`, когда нужна проверка без изменения DNS (`ingress-nginx` версии `1.1.0` и новее);
- включите `http01CertificateSolverBridging`, если сертификаты пути Gateway API выпускаются через HTTP-01, пока публичный DNS ещё указывает на Ingress;
- выберите способ переключения по текущей схеме входа — в подразделах ниже: автоматический балансировщик, ручной балансировщик или `HostPort/HostWithFailover`;
- перед окончательным переключением пользовательского трафика выполните проверку переключения.

Тестирование модуля alb через Ingress-контроллер

Параметр `spec.migrationGateway` ресурса `IngressNginxController` позволяет тестировать обработку запросов от выбранных IP-адресов без изменения DNS.

Запросы из `sourceCIDRs` продолжают поступать через существующую точку входа Ingress-контроллера, но перенаправляются на внутренний объект `Service` целевого экземпляра модуля `alb`. Остальные запросы продолжают поступать в бэкенды, указанные в исходных ресурсах Ingress.



Параметр доступен в модуле `ingress-nginx` версии `1.1.0` и новее.

Несмотря на использование в данном руководстве для миграции на модуль `alb`, параметр `migrationGateway` не привязан к модулю `alb`.

В `serviceRef` можно указать объект `Service`, предоставляющий точку входа любой реализации Gateway API и принимающий HTTP- и HTTPS-трафик на заданных портах.

Найдите конфигурационный объект `Service` модуля `alb`:

```
d8 k get service --all-namespaces \
--selector alb.deckhouse.io/configuration-service
```

Пример вывода для `ClusterALBInstance` с именем `main` — `Service` с типом `ClusterIP` в неймспейсе `d8-alb`:

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
d8-alb	main	ClusterIP	10.222.1.10	<none>	80/TCP,443/TCP	10d

Для `ALBInstance` объект `Service` находится в неймспейсе `ALBInstance`, а его имя — `d8-alb-<ALB_INSTANCE_NAME>`.

Настройте исходный `IngressNginxController`. Сначала укажите в `sourceCIDRs` небольшое количество адресов тестовых клиентов:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <CONTROLLER_NAME>
spec:
  # Поля spec, не связанные с migrationGateway (inlet, ingressClass и т. д.),
  опущены.
  migrationGateway:
    sourceCIDRs:
      - <SOURCE_CIDR>
    serviceRef:
      namespace: <SERVICE_NAMESPACE>
      name: <SERVICE_NAME>
      ports:
        http: <HTTP_PORT>
        https: <HTTPS_PORT>
```

где:

- `<CONTROLLER_NAME>` — имя исходного объекта `IngressNginxController`;
- `<SOURCE_CIDR>` — CIDR клиентов, чьи запросы во время тестирования перенаправляются на модуль `alb`;
- `<SERVICE_NAMESPACE>` — неймспейс конфигурационного Service модуля `alb`;
- `<SERVICE_NAME>` — имя конфигурационного Service модуля `alb`;
- `<HTTP_PORT>` — HTTP-порт целевого Service;
- `<HTTPS_PORT>` — HTTPS-порт целевого Service.

Для выбранных клиентов `migrationGateway` перенаправляет запросы на порты целевого Service и обходит бэкенды и настройки блоков `location` из исходных ресурсов Ingress. HTTP-запросы идут на указанный HTTP-порт. Для HTTPS `nginx` завершает входящее TLS-соединение и устанавливает новое TLS-соединение к указанному HTTPS-порту, используя исходное имя хоста в заголовке `Host` и для SNI. Поддерживаются HTTP, HTTPS и протоколы с HTTP Upgrade, например, WebSocket. gRPC и протоколы, не основанные на HTTP, не поддерживаются.

Параметр действует на все ресурсы Ingress, обслуживаемые контроллером. Перед расширением `sourceCIDRs` проверьте каждое имя хоста, доступное с этих адресов, и убедитесь, что аутентификация, перенаправления, обработка заголовков, GeoIP, соединения WebSocket и другие необходимые политики реализованы в конфигурации модуля `alb`.

Параметр `migrationGateway` работает, перенаправляя на целевой Service уже принятый и, при необходимости, заново зашифрованный HTTP или HTTPS-запрос, поэтому он не поддерживается там, где `nginx` не выполняет эту обработку:

- с инлетами `HostPortWithSSLPassthrough` и `LoadBalancerWithSSLPassthrough` — при сквозной передаче TLS `nginx` не расшифровывает трафик и не может определить, какой запрос нужно перенаправить;
- с инлетом `HostWithFailover` — этот инлет всегда использует Proxy Protocol, а `migrationGateway` не поддерживает передачу Proxy Protocol на целевой Service;
- с параметром `enableIstioSidecar` — трафик обрабатывается сайдкармом Istio, а не напрямую `nginx`.

Целевой Service должен принимать обычный HTTP- и HTTPS-трафик без Proxy Protocol — `migrationGateway` его не передаёт.

Если в целевой конфигурации модуля `alb` нужен Proxy Protocol, на время теста через `migrationGateway` отключите его (`spec.useProxyProtocol: false`). Перед переключением пользовательского трафика снова включите его и проверьте через балансировщик, который передаёт Proxy Protocol.

Если перед Ingress-контроллером используется доверенный L7-балансировщик, проверьте определение исходного IP-адреса клиента, прежде чем использовать `sourceCIDRs` для отбора запросов.

Чтобы немедленно вернуть выбранных клиентов на бэкенды, указанные в исходных ресурсах Ingress, удалите `spec.migrationGateway`:

```
d8 k patch ingressnginxcontroller <CONTROLLER_NAME> --type json \
  --patch ' [{"op": "remove", "path": "/spec/migrationGateway"} ] '
```

где `<CONTROLLER_NAME>` — имя исходного объекта `IngressNginxController`.

Обеспечение проверки HTTP-01 во время миграции

Параметр `migrationGateway` перенаправляет выбранные запросы приложений.

Параметр `migrations.http01CertificateSolverBridging` обеспечивает прохождение проверочных запросов HTTP-01 от cert-manager для Gateway API через Ingress-контроллер, пока публичные DNS-записи указывают на него.

Если используются объекты `Issuer` или `ClusterIssuer` с механизмом проверки HTTP-01, выполните следующие действия:

1. Включите перенаправление проверок до запроса сертификатов для точки входа Gateway API:

```
d8 k patch moduleconfig alb --type merge \
  --patch ' {"spec": {"settings": {"migrations": {"http01CertificateSolverBridging": {"enabled": true, "ingressClassName": "<INGRESS_CLASS>"}}}} '
d8 k -n d8-alb rollout status deployment/gateway-controller
```

где `<INGRESS_CLASS>` — `IngressClass`, через который в данный момент принимается

публичный трафик на порту `80`.

2. Оставьте модуль `ingress-nginx` включённым. Контроллер модуля `alb` создаёт временные ресурсы Ingress для объектов HTTPRoute, создаваемых `cert-manager` для проверки HTTP-01. Благодаря этому запросы проверки проходят через Ingress-контроллер к соответствующему объекту Service.
3. Выпустите сертификаты для точки входа Gateway API до переключения трафика. Проверьте конфигурацию TLS через `migrationGateway` либо подключитесь непосредственно к IP-адресу точки входа Gateway API, переопределив разрешение имени с помощью параметра `curl --resolve`.
4. После переключения DNS или внешнего балансировщика на точку входа Gateway API проверьте выпуск сертификата непосредственно через Gateway API и отключите перенаправление проверок:

```
d8 k patch moduleconfig alb --type merge \
  --patch '{"spec":{"settings":{"migrations":{"http01CertificateSolverBridging":
{"enabled":false}}}}}'
```

Выбор способа переключения

Выберите способ переключения в зависимости от того, как сейчас принимается внешний трафик: через автоматически создаваемый балансировщик, через балансировщик под ручным управлением или напрямую через HostPort/HostWithFailover.

- DNS / автоматически создаваемый LB
- Балансировщик под ручным управлением
- HostPort или HostWithFailover

Автоматически создаваемый балансировщик нагрузки

Чтобы переключить трафик через DNS, выполните следующие действия:

1. Дождитесь готовности балансировщика модуля `alb` и успешного прохождения проверок состояния.
2. Заранее уменьшите TTL DNS.
3. Измените DNS-записи приложений, заменив адрес Ingress-контроллера адресом точки входа Gateway API.

Если Ingress-контроллер использует объект Service типа `LoadBalancer`, автоматически создаваемый облачным провайдером Kubernetes, исходный и целевой контроллеры обычно получают разные IP-адреса или DNS-имена балансировщиков.

Используйте взвешенные DNS-записи для постепенного переключения только при их поддержке DNS-провайдером.

Балансировщик под управлением облачного провайдера обычно не позволяет постепенно переключать отдельные узлы между двумя независимо управляемыми объектами Service.

Если MetalLB или другая реализация использует фиксированный адрес, этот адрес нельзя одновременно назначить двум объектам Service. Переносите его только в рамках согласованного переключения.

Балансировщик нагрузки под ручным управлением

Чтобы переключить трафик через внешний балансировщик, выполните следующие действия:

1. Оставьте публичную DNS-запись без изменений.
2. Добавьте в пул бэкендов внешнего балансировщика узлы модуля `alb` и настроенные для них порты и убедитесь в успешном прохождении проверок состояния.
3. Постепенно переключите трафик с бэкендов Ingress-контроллера.
4. Удаляйте старые бэкенды только после проверки.

Такой способ позволяет переключать отдельные узлы и распределять трафик по весам, если внешний балансировщик поддерживает эти функции.

При настройке бэкендов учитывайте следующее:

- Настройте проверки состояния по пути `/healthz` на HTTP-порту модуля `alb`.
- Сохраните исходный протокол и способ определения IP-адреса клиента: согласуйте настройки Proxy Protocol для пользовательского трафика и проверок состояния либо

- настройте доверие к заголовкам с адресом клиента, которые передаёт L7-балансировщик.
- Если оба контроллера работают на одних узлах, используйте разные порты хоста либо выберите непересекающиеся группы узлов.

Прямой доступ через HostPort или HostWithFailover

Чтобы переключить трафик при прямом доступе через HostPort или HostWithFailover, выполните следующие действия:

- Разверните модуль `alb` на отдельной группе узлов или используйте непересекающиеся порты хоста.
- Для постепенного переключения добавляйте адреса узлов модуля `alb` в DNS-запись и удаляйте адреса узлов Ingress-контроллера после проверки каждого узла.
- Учитывайте TTL DNS и кеширование на стороне клиентов: при указании адресов узлов DNS не обеспечивает предсказуемое весовое распределение трафика и плавное завершение активных соединений.

DNS не позволяет выбрать контроллеры, использующие разные порты на одном адресе узла. Если клиенты должны продолжать использовать порты `80` и `443`, используйте разные узлы либо добавьте балансировщик или правило NAT.

Инлет `HostWithFailover` не имеет прямого аналога в модуле `alb`. Если требуется эквивалентная отказоустойчивость, используйте инлет `LoadBalancer` с MetalLB.

Проверка переключения



Правила обработки трафика в Ingress-контроллере и модуле `alb` могут различаться, в том числе правила формирования и проверки заголовков, обработка протоколов и набор поддерживаемых функций. Перед переключением пользовательского трафика тщательно протестируйте приложения через модуль `alb` и при необходимости адаптируйте их к особенностям его работы.

Перед переключением пользовательского трафика:

- Убедитесь, что ресурсы из шага 3 применены.
- В статусе целевого `ALBInstance` или `ClusterALBInstance` проверьте, что установлены `ready: true` и `synced: true`.
- Проверьте, что поля `conflictPorts`, `conflictBackendTLS` и `conflictFrontendTLS` имеют значение `false`; если какое-то из них `true`, соответствующее поле `<CONFLICT>Owner` укажет имя «управляющего» инстанса.
- Проверьте условия `Accepted` и `Programmed` в статусе `Gateway` и `ListenerSet`, а также `Accepted` в статусе маршрутов (`d8 k -n <NAMESPACE> get gateway,listenerset,httproute <NAME> -o yaml` — раздел `status.conditions`).
- Значение `True` в этих условиях означает, что ресурс принят контроллером и его конфигурация применена.
- Для каждого имени хоста и протокола проверьте обработку трафика, подключившись напрямую к адресу точки входа `Gateway API` или с помощью `migrationGateway: TLS-сертификаты, перенаправления, аутентификацию, долгоживущие соединения и политики приложений`.
- Проверьте сохранение IP-адреса клиента, обработку `Proxy Protocol` или заголовков с адресом клиента, ограничения доступа по источнику и проверки состояния балансировщика.
- Если для HTTP-01 ещё нужно перенаправление проверок, оставьте `http01CertificateSolverBridging` включённым до переключения DNS или балансировщика.
- Убедитесь, что метрики, журналы, оповещения и дашборды позволяют различать трафик и ошибки обоих контроллеров.
- Сохраняйте ресурсы Ingress, Ingress-контроллер, его внешнюю точку входа и действительные сертификаты в течение всего периода, предусмотренного для отката.
- Выберите способ переключения выше и определите критерии отката.

Откат

Определите критерии отката до переключения. Если при проверке выявлены ошибки:

- При тестировании через `migrationGateway` удалите `spec.migrationGateway`.
- При переключении DNS верните адрес Ingress-контроллера и дождитесь истечения предыдущего TTL DNS.
- При использовании балансировщика под ручным управлением верните узлы и порты Ingress-контроллера в его пул бэкендов.

4. Если публичный трафик снова поступает через Ingress-контроллер, а объекты Issuer или ClusterIssuer для сертификатов Gateway API используют механизм проверки HTTP-01, повторно включите `http01CertificateSolverBridging`.

Не удаляйте исходные ресурсы Ingress, Ingress-контроллер, балансировщик или DNS-записи, пока возможность отката не проверена и не завершился согласованный период стабилизации.

Шаг 5. Очистка

После завершения периода отката выполните следующие действия:

1. Удалите `migrationGateway`, если он ещё задан.
2. Отключите `http01CertificateSolverBridging` и `ingress2Gateway`, если они ещё включены.
3. Удалите ресурсы Ingress и Ingress-контроллеры, которые больше не обслуживают приложения и системные интерфейсы DKP.
4. Удалите неиспользуемые балансировщики, сертификаты, объекты Secret и DNS-записи.
5. Верните обычные значения TTL DNS, убедившись, что клиенты больше не используют старую точку входа.

i Часть интерфейсов DKP может по-прежнему публиковаться через Ingress API. Не отключайте модуль `ingress-nginx` и не удаляйте связанные объекты, пока нужные интерфейсы не опубликованы через Gateway API и не проверены. Инструкция находится в разделе «Публикация служебных доменов».

Дополнительные ресурсы

- ALB средствами Kubernetes Gateway API
- ALB средствами Ingress NGINX Controller
- Балансировка входящего трафика
- Документация модуля `alb`

ALB средствами Istio

ALB средствами Istio реализуется через Istio Ingress Gateway или Ingress NGINX. Для этого используется модуль `istio`.

Используйте этот вариант, если требуются расширенные возможности управления трафиком в сервис-меше, например, canary-маршрутизация или mTLS. Подробнее о настройке и возможностях — в документации модуля `istio`.

Создание `IngressIstioController` и подготовка инфраструктуры — задача администратора кластера. Публикация приложения ресурсами Gateway и VirtualService, в том числе canary-развёртывание, описана в разделе «Публикация приложений с использованием ресурса Istio Ingress Gateway».

Ingress для публикации приложений

Приложения можно публиковать средствами Istio Ingress Gateway или через Ingress NGINX с включённым Istio-сайдкаром.

Istio Ingress Gateway

Для публикации приложения средствами Istio Ingress Gateway выполните следующие действия:

1. Создайте ресурс `IngressIstioController`.

В примере ниже создаётся контроллер с инлетом `HostPort` на frontend-узлах:


```

apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  # Значение селектора лейблов, используемое при создании ресурса Gateway.
  ingressGatewayClass: istio-hp
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  resourcesRequests:
    mode: VPA

```

2. Создайте Secret с TLS-сертификатом и ключом для HTTPS. Разместите его в неймспейсе `d8-ingress-istio` и сообщите разработчикам приложения имя Secret и значение `ingressGatewayClass`:

```

apiVersion: v1
kind: Secret
metadata:
  name: app-tls-secret
  namespace: d8-ingress-istio # Неймспейс Secret – d8-ingress-istio, не
неймспейс приложения.
type: kubernetes.io/tls
data:
  tls.crt: |
    <TLS_CERT_DATA>
  tls.key: |
    <TLS_KEY_DATA>

```

Поддерживаемые форматы Secret — в руководстве Istio по форматам ключей.

Далее разработчики приложения создают ресурсы Gateway и VirtualService. Примеры, в том числе canary-развёртывание, — в разделе «Публикация приложений с использованием ресурса Istio Ingress Gateway».

Ingress NGINX

Для публикации через Ingress NGINX с Istio-сайдкаром включите параметр `enableIstioSidecar` в IngressNginxController модуля `ingress-nginx` и сообщите разработчикам приложения имя `ingressClass`.

Манифесты Ingress и Service с обязательными аннотациями — в разделе «Публикация приложений с использованием Ingress NGINX».

Дополнительные ресурсы

- Публикация приложений средствами Istio
- Балансировка входящего трафика
- Документация модуля `istio`

Обеспечение высокой доступности и отказоустойчивости (keepalived)

В Deckhouse Kubernetes Platform для обеспечения высокой доступности и отказоустойчивости можно использовать модуль `keepalived`.

Для настройки keepalived-кластеров на узлах с его помощью используются Custom Resources (кастомные ресурсы).

Примеры использования модуля

Несколько публичных IP-адресов

Рассмотрим пример, когда три публичных IP-адреса расположены на трех front-узлах. Каждый виртуальный IP-адрес привязан к отдельной VRRP-группе, что обеспечивает независимое перемещение каждого адреса. При наличии в кластере трёх узлов с лейблами `node-role.deckhouse.io/frontend: ""`, каждый IP-адрес будет закреплён за своим master-узлом.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Обязательно.
    node-role.deckhouse.io/frontend: ""
  tolerations: # Опционально.
    - key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  vrrpInstances:
    - id: 1 # Уникальный для всего кластера ID.
      interface:
        detectionStrategy: DefaultRoute # В качестве служебной сетевой карты
        # используется та, через которую проложен дефолтный маршрут.
      virtualIPAddresses:
        - address: 42.43.44.101/32
          # В приведённом примере IP-адреса переходят по тем же сетевым интерфейсам, по
          # которым передаётся служебный VRRP-трафик, поэтому параметр interface указывать не
          # требуется.
    - id: 2
      interface:
        detectionStrategy: DefaultRoute
      virtualIPAddresses:
        - address: 42.43.44.102/32
    - id: 3
      interface:
        detectionStrategy: DefaultRoute
      virtualIPAddresses:
        - address: 42.43.44.103/32
```

Шлюз использует пару IP-адресов — для LAN и WAN. В отличие от других случаев, приватный и публичный IP-адреса связаны между собой и перемещаются между узлами совместно. В данном примере служебный VRRP-трафик передаётся через LAN-интерфейс, который определяется методом `NetworkAddress` (предполагается, что на каждом узле имеется IP-адрес из соответствующей подсети).

```

apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
    - key: node-role.deckhouse.io/mygateway
      operator: Exists
  vrrpInstances:
    - id: 4 # ID "1", "2", "3" уже используются в KeepalivedInstance "front" выше.
      interface:
        detectionStrategy: NetworkAddress
        networkAddress: 192.168.42.0/24
      virtualIPAddresses:
        - address: 192.168.42.1/24
          # Так как локальная сеть уже определена выше, параметр interface для этого IP
          # можно не указывать.
        - address: 42.43.44.1/28
          interface:
            detectionStrategy: Name
            name: ens7 # Интерфейс для публичных IP на всех узлах называется "ens7",
            # указываем его явно.

```

Ручное переключение keepalived

1. Зайдите в нужный под, используя debug-контейнер с общим пространством процессов:

```
d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived
```

2. Отредактируйте файл конфигурации `vim /proc/1/root/etc/keepalived/keepalived.conf`, где в строке с параметром `priority` замените значение на `<число подов keepalived + 1>` или установите значение выше, чем у текущего VRRP-мастера (например, `255`).
3. Примените настройки – отправьте сигнал на перечитывание конфигурации::

```
kill -HUP 1
```

Дополнительные ресурсы

- Документация модуля `keepalived`

Настройка Egress Gateway



Функция доступна в DKP Standard Edition+, Enterprise Edition, SE Lite (1.73), CSE Pro (1.73).

Egress Gateway позволяет централизованно управлять исходящим трафиком и контролировать его, а также обеспечивает такие функции, как шифрование, маршрутизация и мониторинг.



Чтобы использовать Egress Gateway, включите в кластере модуль `cni-cilium`.

Режимы использования Egress Gateway

Базовый режим

Используются предварительно настроенные IP-адреса на egress-узлах.

Режим с Virtual IP

Реализована возможность динамически назначать дополнительные IP-адреса узлам.

Принцип настройки

Для настройки egress-шлюза добавьте два кастомных ресурса:

1. `EgressGateway` — определяет группу узлов, один из которых будет выбран в качестве активного egress-шлюза, а остальные останутся в резерве на случай отказа:
 - Из узлов, попадающих под `spec.nodeSelector`, будут отобраны пригодные к использованию. Один из них будет назначен активным шлюзом. Выбор активного узла осуществляется в алфавитном порядке.

Узел считается пригодным, если:

 - находится в состоянии `Ready`.
 - не помечен как недоступный (`cordon`).
 - агент `cilium-agent` на нём также в состоянии `Ready`.
 - Если `EgressGateway` используется в режиме `VirtualIP`, на активном узле запускается агент, который эмулирует виртуальный IP-адрес с использованием протокола ARP. При определении пригодности узла также учитывается состояние пода данного агента.
 - Один и тот же узел может быть задействован в нескольких `EgressGateway`. Выбор активного узла в каждом `EgressGateway` осуществляется независимо от других, что позволяет сбалансировать нагрузку между ними.
2. `EgressGatewayPolicy` — определяет политику перенаправления сетевых запросов от подов в кластере на конкретный egress-шлюз, заданный с помощью `EgressGateway`.

Обслуживание узла

Для проведения работ на узле, который в данный момент является активным egress-шлюзом, выполните следующие шаги:

1. Снимите лейбл с узла, чтобы исключить его из списка кандидатов для роли egress-шлюза. `<egress-label>` — это лейбл, указанный в `spec.nodeSelector` вашего `EgressGateway`.

```
d8 k label node <node-name> <egress-label>-
```

2. Переведите узел в режим обслуживания (`cordon`), чтобы предотвратить запуск новых подов:

```
d8 k cordon <node-name>
```

После этого Cilium автоматически выберет новый активный узел из оставшихся кандидатов. Трафик продолжит направляться через новый шлюз без прерывания.

3. Для возврата узла в работу выполните:

```
d8 k uncordon <node-name>
d8 k label node <node-name> <egress-label>=<value>
```



Повторное добавление лейбла может привести к тому, что узел снова будет выбран активным egress-шлюзом (если он первый в алфавитном порядке среди доступных кандидатов).

Чтобы избежать немедленного возврата узла в активное состояние, временно уменьшите количество реплик в `EgressGateway` или настройте приоритет выбора через дополнительные лейблы.

Сравнение с `CiliumEgressGatewayPolicy`

`CiliumEgressGatewayPolicy` подразумевает настройку только одного узла в качестве egress-шлюза. При выходе его из строя не предусмотрено failover-механизмов и сетевая связь будет нарушена.

Примеры настройки

EgressGateway в режиме `PrimaryIPFromEgressGatewayNodeInterface`

(базовый режим)

```

apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: myegressgw
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # Все узлы, подходящие под nodeSelector, должны иметь одинаковое имя публичного
      # интерфейса.
      # При выходе из строя активного узла, трафик будет перенаправлен через
      # резервный.
      # при этом IP-адрес источника в сетевых пакетах изменится.
      interfaceName: eth1

```

EgressGateway в режиме VirtualIPAddress (Режим с Virtual IP)

```

apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: myeg
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # На каждом узле должны быть настроены все необходимые маршруты для доступа на
      # все внешние публичные сервисы,
      # Публичный интерфейс должен быть подготовлен к автоматической настройке
      # виртуального IP в качестве дополнительного (secondary) IP-адреса.
      # При выходе из строя активного узла, трафик будет перенаправлен через
      # резервный.
      # IP-адрес отправителя у сетевых пакетов не поменяется.
      ip: 172.18.18.242
      # Список сетевых интерфейсов, на которых будет назначен виртуальный IP-адрес.
      interfaces:
        - eth1

```

EgressGatewayPolicy

```

apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
    - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
    - podSelector:
        matchLabels:
          app: backend
          io.kubernetes.pod.namespace: my-ns

```

Дополнительные ресурсы

- Параметры модуля `cni-cilium`
- Кастомные ресурсы модуля `cni-cilium`

Межкластерное взаимодействие (альянс кластеров)

В Deckhouse Kubernetes Platform (DKP) поддерживаются две схемы межкластерного взаимодействия:

- Мультикластер;
- Федерация.

Обе схемы могут быть реализованы средствами Istio (модуль `istio`) или Cilium (модуль `cni-cilium`). Оба инструмента обеспечивают развёртывание service mesh (далее — сервис-меш) для контроля и настройки сетевых взаимодействий между приложениями внутри кластера.

Особенности использования service mesh DKP

Доступные сценарии зависят от редакции платформы.

Мультикластер и федерация доступны в редакциях Enterprise Edition (EE) и Certified Security Edition Pro (CSE Pro).

В редакциях Community Edition (CE), Basic Edition (BE), Standard Edition (SE), Standard Edition+ (SE+) и Certified Security Edition Lite (CSE Lite) можно применять сервис-меш только в рамках одного кластера, например для реализации паттернов Circuit Breaker, Canary Deployment и т.д.

Сценарий использования	CE / BE / SE / SE+ / CSE Lite	EE	CSE Pro
Сервис-меш в одном кластере	+	+	+
Федерация	-	+	+
Мультикластер	-	+	+

Отличия федерации и мультикластера

Федерация объединяет независимые (суверенные) кластеры:

- У каждого кластера собственное пространство имен и сервисов.
- Взаимодействие между кластерами настраивается явно — указывается, какие сервисы доступны извне.

Мультикластер объединяет тесно связанные кластеры:

- Используется общее пространство имён.
- Сервисы из одного кластера доступны для других, как если бы они были локальными (если это не ограничено политиками авторизации).

Модели объединения нескольких кластеров (на примере Istio)

При использовании Istio возможны различные модели объединения:

- Сетевые модели: кластеры находятся в одной или в разных сетях.
- Модели использования управляющего слоя: один или несколько управляющих слоев service mesh.
- Модели изоляции: изоляция tenants на уровне пространств имен, сетей или кластеров.
- Модели использования сервис-мешей: один сервис-меш или несколько.

Эти параметры можно комбинировать, чтобы построить архитектуру, соответствующую требованиям конкретной инфраструктуры.

Задачи, решаемые с помощью мультикластера и федерации

Объединение кластеров в федерацию или мультикластер поможет решить следующие задачи:

- Геораспределенность и ГЕО-балансировка трафика: кластеры обслуживают пользователей ближе к их локации.
- Отказоустойчивость: при сбое работы кластера или дата-центра, трафик автоматически переключается на другие (автоматический failover).

- Горизонтальное масштабирование: распределение нагрузки между кластерами увеличивает общую производительность системы.
- Изоляция окружений и ресурсов: разные среды (production, staging, dev) могут размещаться в отдельных кластерах, при необходимости — с возможностью взаимодействия.
- Соответствие требованиям регуляторов: например, по хранению данных в определённых географических зонах.
- Создание гибридных и мульти-облачных инфраструктур: объединение кластеров, размещённых в разных средах (облако, разные провайдеры и т.д.).

Дополнительные ресурсы

- Документация модуля `cni-cilium`
- Документация модуля `istio`

Мультикластер

Мультикластер средствами Istio

i Доступно только в DKP Enterprise Edition (EE) и DKP Certified Security Edition Pro (CSE Pro).

Требования к кластерам

- У всех кластеров, входящих в мультикластер, значение параметра `clusterDomain` ресурса `ClusterConfiguration` должно совпадать. По умолчанию используется значение `cluster.local`.
- Подсети сервисов и подов в параметрах `serviceSubnetCIDR` и `podSubnetCIDR` ресурса `ClusterConfiguration` должны быть уникальными для каждого кластера.

- i**
- При анализе HTTP и HTTPS запросов (*в терминологии Istio*) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно по заголовкам.
 - А при анализе TCP запросов (*в терминологии Istio*) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно только по IP-адресу назначения и номеру порта.

Если IP-адреса сервисов или подов пересекутся между кластерами, то под маршрутизирующие, запрещающие или разрешающие правила Istio могут попасть запросы из подов других кластеров. Пересечение подсетей сервисов и подов не рекомендуется (подробнее — в документации Istio).

Istio работает в режиме multi-network — поды разных кластеров взаимодействуют друг с другом только через Istio ingress gateway. Прямое взаимодействие между подами разных кластеров не поддерживается.

Общие принципы

- Для работы мультикластера требуется взаимное доверие между кластерами. Это реализуется через обмен корневыми сертификатами между кластерами.
- Для сбора информации о соседних сервисах Istio подключается к API-серверу соседнего кластера. Настройку защищённого канала обеспечивает модуль `istio` в составе Deckhouse.

Включение мультикластера

При включении мультикластера (параметр модуля `istio.multicluster.enabled = true`) происходит следующее:

- В кластер добавляется прокси для публикации доступа к API-серверу через стандартный Ingress-ресурс:
 - Доступ защищён авторизацией на основе Bearer-токенов, подписанных доверенными ключами.
 - Обмен доверенными публичными ключами происходит автоматически средствами Deckhouse при взаимной настройке мультикластера.
 - Прокси имеет read-only-доступ к строго определённым ресурсам.
- В кластер добавляется сервис, который экспортирует метаданные кластера наружу:
 - Корневой сертификат Istio (доступен без аутентификации).

- Публичный адрес, через который доступен API-сервер (доступен только для аутентифицированных запросов из соседних кластеров).
- Список публичных адресов сервиса `ingressgateway` (доступен только для аутентифицированных запросов из соседних кластеров).
- Публичные ключи сервера для аутентификации запросов к API-серверу и закрытым метаданным.

Управление мультикластером

Для сборки мультикластера необходимо в каждом кластере создать набор ресурсов `IstioMulticluster`, описывающих остальные кластеры.

Пример устройства мультикластера из двух кластеров

Для устройства мультикластера средствами Istio используйте кастомный ресурс `IstioMulticluster`.

Кластер A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Кластер B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `istio`

Федерация

Федерация средствами Istio (Service Mesh)

i Доступно только в DKP Enterprise Edition (EE) и DKP Certified Security Edition Pro (CSE Pro).

Требования к кластерам

- Каждый кластер должен иметь уникальное значение параметра `clusterDomain` в ресурсе `ClusterConfiguration`. Обратите внимание, что ни один из кластеров не должен иметь домен `cluster.local`, который является значением по умолчанию.

i Значение `cluster.local` использовать нельзя, так как это зарезервированный псевдоним для домена локального кластера. Если в `AuthorizationPolicy` указать `cluster.local` как `principals`, правило будет применяться только к локальному кластеру, даже если в Service mesh существует кластер, у которого `clusterDomain` явно определен как `cluster.local` (Подробнее — в документации Istio).

- Требования к уникальности подсетей сервисов и подов в параметрах `serviceSubnetCIDR` и `podSubnetCIDR` ресурса `ClusterConfiguration` при работе кластеров в федерации отсутствуют.



При анализе трафика Istio использует:

- для HTTP/HTTPS-запросов — заголовки;
- для TCP-запросов — IP-адрес назначения и порт.

Istio работает в режиме multi-network — поды разных кластеров взаимодействуют друг с другом только через Istio ingress gateway. Прямое взаимодействие между подами разных кластеров не поддерживается.

Общие принципы федерации

- Федерация требует взаимного доверия между кластерами. Для этого необходимо обменяться корневыми сертификатами: кластер А должен доверять кластеру В и наоборот.
- Для настройки межкластерного доступа к сервисам необходимо обменяться информацией о публичных сервисах. Чтобы опубликовать сервис bar из кластера Б в кластере А, необходимо в кластере А создать ресурс ServiceEntry, который описывает публичный адрес ingress-gateway кластера Б.

Включение федерации: создаваемые сервисы

При включении федерации (установка параметра модуля `istio.federation.enabled = true`) в кластер добавляются:

- Сервис `ingressgateway`, который проксирует mTLS-трафик извне к прикладным сервисам.
- Сервис экспорта метаданных:
 - корневой сертификат Istio (доступен без аутентификации);
 - список публичных сервисов в кластере (доступен только для аутентифицированных запросов из соседних кластеров);
 - список публичных адресов сервиса `ingressgateway` (доступен только для аутентифицированных запросов из соседних кластеров).

Настройка федерации

Для настройки федерации необходимо:

- В каждом кластере создать набор ресурсов IstioFederation для описания других кластеров.
 - После успешного автосогласования между кластерами в ресурсе IstioFederation будут записаны необходимые служебные данные в `status.metadataCache.public` и `status.metadataCache.private`.
- Каждый ресурс (Service), который считается публичным в рамках федерации, пометить лейблом `federation.istio.deckhouse.io/public-service: ""`.
 - В остальных кластерах из состава федерации, для каждого ресурса Service создадутся соответствующие ServiceEntry, указывающие на `ingressgateway` исходного кластера.



Важно. Убедитесь, что поле `name` в разделе `.spec.ports` ресурса Service заполнено для каждого порта, иначе могут быть проблемы в работе федерации.

Пример устройства федерации из двух кластеров

Для настройки федерации средствами Istio используйте кастомный ресурс IstioFederation.

Кластер А:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Кластер В:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Кастомные ресурсы модуля `istio`

Настройка сетевых политик

Если на уровне инфраструктуры, где работает Deckhouse Kubernetes Platform, есть требования для ограничения сетевого взаимодействия между серверами, то необходимо соблюсти следующие условия:

- Включен режим туннелирования трафика между подами (настройки для CNI Cilium, настройки для CNI Flannel).
- Разрешена передача трафика между `podSubnetCIDR`, инкапсулированного внутри VXLAN (если выполняется инспектирование и фильтрация трафика внутри VXLAN-туннеля).
- В случае необходимости интеграции с внешними системами (например, LDAP, SMTP или прочие внешние API), с ними разрешено сетевое взаимодействие.
- Локальное сетевое взаимодействие полностью разрешено в рамках каждого отдельно взятого узла кластера.
- Разрешено взаимодействие между узлами по портам, приведенным в списке сетевого взаимодействия компонентов платформы. Обратите внимание, что большинство портов входит в диапазон 4200-4299. При добавлении новых компонентов платформы им будут назначаться порты из этого диапазона (при наличии возможности).

Настройка сетевых политик стандартными средствами Kubernetes

Пример настроек сетевой политики

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              project: myproject
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
          port: 5978

```

Настройка сетевых политик на уровне всего кластера с помощью CiliumClusterwideNetworkPolicy

Для определения сетевых политик на уровне всего кластера в Deckhouse Kubernetes Platform можно использовать объекты CiliumClusterwideNetworkPolicy модуля `cni-cilium`.



Использование объектов CiliumClusterwideNetworkPolicy без включения параметра `policyAuditMode` в настройках модуля `cni-cilium` может привести к некорректной работе control plane или потере доступа ко всем узлам кластера по SSH.

Для использования объектов CiliumClusterwideNetworkPolicy выполните следующие шаги:

1. Примените первичный набор объектов CiliumClusterwideNetworkPolicy. Для этого в настройках модуля `cni-cilium` добавьте конфигурационную опцию `policyAuditMode` со значением `true`.

Опция `policyAuditMode` может быть удалена после применения всех объектов CiliumClusterwideNetworkPolicy и проверки корректности их работы в Hubble UI.

2. Примените правило политики сетевой безопасности:

```
apiVersion: "cilium.io/v2"
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: "allow-control-plane-connectivity"
spec:
  ingress:
    - fromEntities:
      - kube-apiserver
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/control-plane: ""
```

В случае, если объекты `CiliumClusterwideNetworkPolicy` не применяются, control plane может некорректно работать до одной минуты во время перезагрузки подов `cilium-agent`. Это происходит из-за сброса Conntrack-таблицы. Привязка к entity `kube-apiserver` позволяет избежать проблемы.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `cni-cilium`
- Параметры модуля `cni-flannel`

Программно-определяемые сети (SDN)

В Deckhouse Kubernetes Platform (DKP) поддерживается добавление в кластер дополнительных программно-определяемых сетей (SDN, Software-Defined Networking). Функции дополнительных программно-определяемых сетей (далее — дополнительные сети) в рамках DKP реализуются с помощью модуля `sdn`.

Программно-определяемые сети в DKP позволяют декларативно управлять дополнительными сетевыми сегментами для прикладных нагрузок (поды, виртуальные машины). Вместо ручной настройки сетевых интерфейсов на каждом узле кластера, администратор описывает желаемое состояние сети через кастомные ресурсы Kubernetes, а модуль `sdn` автоматически настраивает необходимое сетевое оборудование.

DKP поддерживает следующие возможности по работе с программно-определяемыми сетями:

- Конфигурация сетевых интерфейсов на узлах. Поддерживаются такие функции, как агрегация портов, объединение сетевых интерфейсов в бридж, настройка VLAN-интерфейсов.
- Дополнительные сети для прикладных нагрузок. Поддерживается добавление в кластер дополнительных программно-определяемых сетей: общедоступных в каждом проекте (кластерные), а также доступных в рамках одного неймспейса (сети проекта).
- IPAM для дополнительных сетей. Механизм позволяет выделять IP-адреса (поддерживаются IPv4-адреса) из пулов и назначать их на дополнительные сетевые интерфейсы подов, подключаемых к кластерным сетям и сетям проекта.
- Underlay-сети для проброса аппаратных устройств. Это позволяет приложениям DPDK и другим высокопроизводительным рабочим нагрузкам получать прямой доступ к физическим сетевым интерфейсам (PF/VF), обходя сетевой стек ядра.
- Системные сети (сервисные сети на узлах). Предназначены для служебного трафика на уровне узлов и не используются как дополнительные сети подов. Создаются поверх существующих underlay-сетей.

Дополнительные ресурсы

- Документация модуля `sdn`

Настройка SDN в кластере

Для использования SDN в кластере DKP требуется подготовка инфраструктуры к включению модуля `sdn`, а также некоторые подготовительные действия после его включения.

Подготовка инфраструктуры к включению модуля `sdn`

Перед использованием дополнительных программно-определяемых сетей (далее — дополнительные сети) в кластере требуется предварительная подготовка инфраструктуры:

- Для создания дополнительных программно-определяемых сетей на основе

тегированных VLAN:

- Выделите диапазоны VLAN ID на коммутаторах в ЦОД и настройте их на соответствующих сетевых интерфейсах коммутаторов.
- Выберите физические интерфейсы на узлах для последующей настройки тегированных VLAN-интерфейсов. Допускается использование интерфейсов, уже задействованных для служебной межузловой сети DKP.
- **Для создания дополнительных программно-определяемых сетей на основе прямого нетегированного доступа через сетевой интерфейс:**
 - Выделите отдельные физические интерфейсы на узлах и объедините их в единую сеть на уровне ЦОД.

Включение модуля sdn

Включите модуль `sdn` согласно инструкции.

Действия после включения модуля sdn

После включения модуля в кластере автоматически будут созданы объекты `NodeNetworkInterface`, которые отражают текущее состояние сетевых интерфейсов на узлах.

Проверьте их наличие командой:

```
d8 k get nodenetworkinterface
```

NAME	MANAGEDBY	NODE	TYPE	IFNAME	IFINDEX	STATE
AGE						
virtlab-ap-0-nic-1c61b4a68c2a	Deckhouse	virtlab-ap-0	NIC	eth1	3	Up
35d						
virtlab-ap-0-nic-fc34970f5d1f	Deckhouse	virtlab-ap-0	NIC	eth0	2	Up
35d						
virtlab-ap-1-nic-1c61b4a6a0e7	Deckhouse	virtlab-ap-1	NIC	eth1	3	Up
35d						
virtlab-ap-1-nic-fc34970f5c8e	Deckhouse	virtlab-ap-1	NIC	eth0	2	Up
35d						
virtlab-ap-2-nic-1c61b4a6800c	Deckhouse	virtlab-ap-2	NIC	eth1	3	Up
35d						
virtlab-ap-2-nic-fc34970e7ddb	Deckhouse	virtlab-ap-2	NIC	eth0	2	Up
35d						

i

При обнаружении интерфейсов узлов контроллер устанавливает на них следующие служебные лейблы (пример):

```
labels:
  network.deckhouse.io/interface-mac-address: fa163eebea7b
  network.deckhouse.io/interface-type: NIC
  network.deckhouse.io/nic-pci-bus-info: 0000-17-00.0
  network.deckhouse.io/nic-pci-type: PF
  network.deckhouse.io/node-name: worker-01
annotations:
  network.deckhouse.io/heritage: NetworkController
```

В примере выше у каждого узла в кластере есть по два сетевых интерфейса — `eth0` (DKP LAN) и `eth1` (выделенный интерфейс для организации дополнительных программно-определяемых сетей).

Разметка интерфейсов для организации дополнительных программно-определяемых сетей

Чтобы использовать дополнительные программно-определяемые сети, назначьте выделенным интерфейсам (в примере выше, `eth1`) подходящий лейбл.

Пример:

```
d8 k label nodenetworkinterface virtlab-ap-0-nic-1c61b4a68c2a nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-1-nic-1c61b4a6a0e7 nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-2-nic-1c61b4a6800c nic-group=extra
```

Объединение нескольких физических интерфейсов в интерфейс

агрегации каналов (bond-интерфейс)

Для увеличения пропускной способности или резервирования можно объединить несколько физических интерфейсов в агрегированный канал (интерфейс агрегации каналов, bond-интерфейс).

i Объединять можно только сетевые интерфейсы, расположенные на одном физическом или виртуальном хосте.

Пример создания интерфейса агрегации каналов:

1. Установите на интерфейсы, предназначенные для агрегации, пользовательские лейблы.

Пример установки лейбла `nni.example.com/bond-group=bond0` на интерфейсы:

```
d8 k label nni node-0-nic-fa163efbde48 nni.example.com/bond-group=bond0
d8 k label nni node-0-nic-fa40asdxzx78 nni.example.com/bond-group=bond0
```

2. Подготовьте и примените конфигурацию для создания интерфейса агрегации.

Пример конфигурации:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterface
metadata:
  name: nni-node-0-bond0
spec:
  nodeName: node-0
  type: Bond
  heritage: Manual
  bond:
    bondName: bond0
    memberNetworkInterfaces:
      - labelSelector:
          matchLabels:
            # Служебный лейбл, который необходимо указывать для объединения с
            # интерфейсом Bond на определенном узле.
            network.deckhouse.io/node-name: worker-01
            # Пользовательский лейбл, (был добавлен на интерфейсы на предыдущем
            # шаге).
            nni.example.com/bond-group: bond0
```

3. Проверьте статус созданного интерфейса агрегации.

Получите список интерфейсов:

```
d8 k get nni
```

Пример вывода:

NAME	MANAGEDBY	NODE	TYPE
IFNAME	IFINDEX	STATE	AGE
nni-node-0-bond0	Manual	node-0-b23d3a26-5fb4b-5s9fp	Bond
bond0	76	Up	7m48s
...			

Посмотрите информацию о статусе нужного интерфейса:

```
d8 k get nni nni-node-0-bond0 -o yaml
```

Пример статуса интерфейса:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterface
metadata:
  ...
status:
  conditions:
    - lastProbeTime: "2025-09-30T09:00:54Z"
      lastTransitionTime: "2025-09-30T09:00:39Z"
      message: Interface created
      reason: Created
      status: "True"
      type: Exists
    - lastProbeTime: "2025-09-30T09:00:54Z"
      lastTransitionTime: "2025-09-30T09:00:39Z"
      message: Interface is up and ready to send packets
      reason: Up
      status: "True"
      type: Operational
  deviceMAC: 6a:c7:ab:2a:a6:1e
  groupedLinks:
    - deviceMAC: fa:16:3e:92:14:40
      type: NIC
  ifIndex: 76
  ifName: bond0
  managedBy: Manual
  operationalState: Up
  permanentMAC: ""

```

Настройка и подключение дополнительных виртуальных сетей для использования в прикладных подах

В Deckhouse Kubernetes Platform можно декларативно управлять дополнительными сетями для прикладных нагрузок (поды, виртуальные машины). При этом предусмотрено следующее:

- Каждая дополнительная сеть подразумевает единственный L2-домен обмена данными.
- Внутри сетевого пространства пода дополнительная сеть представлена в виде tap-интерфейса.
- В качестве технологии организации L2-сети в настоящее время поддерживаются следующие режимы:
 - **Тегированный VLAN** — для связи между подами на разных узлах сетевые пакеты помечаются соответствующим VLAN ID и используют инфраструктурное сетевое оборудование для коммутации. Этот метод позволяет создать 4096 дополнительных сетей в рамках одного кластера;
 - **Прямой доступ в сетевой интерфейс на узлах** — для связи между подами на разных узлах используются локальные сетевые интерфейсы на узлах.
- По модели управления сети бывают двух типов:
 - **Кластерная сеть** — сеть, общедоступная в каждом проекте, управляется администратором. Пример — публичная WAN-сеть или shared-сеть обмена трафиком между проектами;
 - **Сеть проекта** — сеть, доступная в рамках неймспейса, управляется пользователем.

Для настройки и подключения дополнительных сетей для прикладных подов используются кастомные ресурсы ClusterNetwork, Network и NetworkClass.



Если в ресурсах Network или ClusterNetwork был указан тип VLAN, также создадутся NodeNetworkInterface для VLAN и Bridge-интерфейсов.



Перед созданием дополнительной сети разметьте интерфейсы, которые будут использоваться для её подключения.

Создание общедоступной (кластерной) сети

Для создания общедоступных сетей в масштабе всего кластера используется кастомный ресурс ClusterNetwork.

Создание общедоступной сети, основанной на тегированном трафике

Чтобы создать сеть, основанную на тегированном трафике, выполните следующие шаги:

1. Создайте и примените ресурс ClusterNetwork.

В параметре `spec.type` укажите значение `VLAN`. На соответствующих сетевых интерфейсах узлов будут настроены тегированные интерфейсы для обеспечения связности через VLAN, предоставленный инфраструктурой.

Пример манифеста ClusterNetwork для создания общедоступной сети, основанной на тегированном трафике:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: VLAN
  vlan:
    id: 900
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        # Лейбл, установленный на ресурсы NodeNetworkInterface на этапе разметки
        # интерфейсов под организацию дополнительных программно-определяемых сетей.
        nic-group: extra
```

2. Проверьте состояние созданного ресурса командой:

```
d8 k get clusternetworks.network.deckhouse.io my-cluster-network -O yaml
```

Пример статуса ресурса ClusterNetwork:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  ...
status:
  bridgeName: d8-br-900
  conditions:
    - lastTransitionTime: "2025-09-29T14:39:20Z"
      message: All node interface attachments are ready
      reason: AllNodeInterfaceAttachmentsAreReady
      status: "True"
      type: AllNodeAttachmentsAreReady
    - lastTransitionTime: "2025-09-29T14:39:20Z"
      message: Network is operational
      reason: NetworkReady
      status: "True"
      type: Ready
  nodeAttachmentsCount: 1
  observedGeneration: 1
  readyNodeAttachmentsCount: 1
```

3. Проверьте присоединение дополнительной сети к интерфейсам на узлах.

Создание сети, основанной на прямом доступе в интерфейс

Для создания сети на основе прямого доступа к интерфейсу используйте ресурс ClusterNetwork.

В параметре `spec.type` укажите значение `Access`. Выбранные сетевые адаптеры на узлах будут использоваться напрямую для обеспечения связности.

Пример манифеста для сети, основанной на прямом доступе в интерфейс:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: Access
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        # Лейбл, установленный на ресурсы NodeNetworkInterface на этапе разметки
        # интерфейсов под организацию дополнительных программно-определяемых сетей.
        nic-group: extra
```

После создания сети проверьте ее присоединение к интерфейсам на узлах.

Создание сети проекта (пользовательской сети)

Чтобы пользователи имели возможность создавать собственные выделенные сети, основанные на тегированном трафике, необходимо предварительно описать доступный им диапазон тегов и определить сетевые интерфейсы, на которых они могут быть настроены. Для этого используется кастомный административный ресурс `NetworkClass`.

Пример:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NetworkClass
metadata:
  name: my-network-class
spec:
  vlan:
    idPool:
      - 600-800
      - 1200
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        nic-group: extra # Лейбл, установленный на ресурсы NodeNetworkInterface на
        # этапе разметки интерфейсов под организацию дополнительных программно-определяемых
        # сетей.
```

По запросу пользователя администратор предоставляет ему название созданного `NetworkClass`, который используется при создании сети проекта.

Пример создания пользовательской сети с использованием созданного административного ресурса `NetworkClass` описан в разделе «Создание сети проекта (пользовательской сети)».

Проверка присоединения дополнительной сети к интерфейсам на узлах

После создания ресурса `ClusterNetwork` или `Network` будет автоматически создан ресурс `NodeNetworkInterfaceAttachment`, отражающий присоединение данной сети к интерфейсам на узлах.

Для получения списка ресурсов `NodeNetworkInterfaceAttachment` и информации о конкретном ресурсе используйте команды:

```
d8 k get nnia
d8 k get nnia my-cluster-network-... -O yaml
```

Пример ресурса `NodeNetworkInterfaceAttachment`:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterfaceAttachment
metadata:
  ...
  finalizers:
    - network.deckhouse.io/nni-network-interface-attachment
    - network.deckhouse.io/pod-network-interface-attachment
  generation: 1
  name: my-cluster-network-...
  ...
spec:
  networkRef:
    kind: ClusterNetwork
    name: my-cluster-network
  parentNetworkInterfaceRef:
    name: right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b
    type: VLAN
status:
  bridgeNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-bridge-900
  conditions:
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Vlan created
      reason: VLANCreated
      status: "True"
      type: Exist
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Bridged successfully
      reason: VLANBridged
      status: "True"
      type: Ready
  nodeName: right-worker-b23d3a26-5fb4b-h2bkv
  vlanNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc

```

Статус `NodeNetworkInterfaceAttachment` изменится на `True` сразу после того как соответствующие `NodeNetworkInterface` появятся и перейдут в состояние `Up`.

Для проверки статусов `NodeNetworkInterface` используйте команду:

```
d8 k get nni
```

Пример вывода:

NAME					MANAGEDBY	NODE
TYPE	IFNAME	IFINDEX	STATE	AGE		
...						
right-worker-b23d3a26-5fb4b-h2bkv-bridge-900					Deckhouse	right-worker-
b23d3a26-5fb4b-h2bkv	Bridge	d8-br-900	684		Up	14h
right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b					Deckhouse	right-worker-
b23d3a26-5fb4b-h2bkv	NIC	ens3	2		Up	19d
right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc					Deckhouse	right-worker-
b23d3a26-5fb4b-h2bkv	VLAN	ens3.900	683		Up	14h
...						

IPAM для дополнительных сетей

Механизм IPAM (IP Address Management) позволяет выделять IPv4-адреса из пулов и назначать их на дополнительные сетевые интерфейсы подов, подключаемых к кластерным сетям и сетям проекта.

Принципы и особенности работы IPAM в DKP

Для каждого выделяемого IP-адреса создаётся и используется объект `IPAddress` (`ClusterIPAddress`

— для кластерных сетей), который ссылается на сеть проекта или кластерную сеть. Контроллер выделяет адрес из пула и сохраняет результат в `status.address`, `status.network`, `status.routes` объекта `IPAddress` (`ClusterIPAddress`). Агент на узле назначает IP-адрес и маршруты на интерфейс внутри пода и устанавливает поля `status.conditions[Attached]` и `status.usedByPods` объекта `IPAddress` (`ClusterIPAddress`).

Защита от конфликтов при переиспользовании IP-адресов

Для защиты от конфликтов создаётся cluster-scoped объект `IPAddressLease`, который резервирует IP-адрес. При удалении объекта `IPAddress` (`ClusterIPAddress`) соответствующий ему `IPAddressLease` помечается как `orphaned` (для этого используется поле `status.orphaningTimestamp`) и удерживает адрес в течение времени, указанном в параметре `spec.ttl` (чтобы избежать быстрых переиспользований IP-адресов).

Ресурсы и параметры для настройки IPAM

Для управления выделением и назначением IP-адресов используются:

- Пулы адресов: для кластерных сетей (ресурс `ClusterIPAddressPool`) или сетей проекта (ресурс `IPAddressPool`).
- Параметры для включения IPAM для конкретной сети:
`Network.spec.ipam.ipAddressPoolRef` — для сетей проекта,
`ClusterNetwork.spec.ipam.ipAddressPoolRef` — для кластерных сетей.
- Ресурс `IPAddress` (`ClusterIPAddress` — для кластерных сетей) — запрос (автоматический или статический) на выделение адреса, который затем назначается на интерфейс пода, подключаемого к кластерной сети или сети проекта.

Пример выделения пула IP-адресов для кластерной сети

i Чтобы выделить пул адресов для кластерной сети, используйте ресурс `ClusterIPAddressPool`.

Для выделения пула адресов, предназначенных для назначения на сетевые интерфейсы подов, подключаемых к кластерной сети, выполните следующие действия:

1. Создайте пул адресов. Для этого используйте ресурс `ClusterIPAddressPool`.

Пример:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterIPAddressPool
metadata:
  name: public-net-pool
spec:
  leaseTTL: 24h
  pools:
    - network: 203.0.113.0/24
      ranges:
        - 203.0.113.10-203.0.113.200
```

i Параметр `spec.pools[].ranges` опционален. Если он не указан, доступным считается весь CIDR из `spec.pools[].network` (за исключением `network/broadcast` адресов, см. поведение `/31` и `/32`).

2. Включите IPAM в сети. Для этого в параметре `spec.ipam.ipAddressPoolRef` ресурса `ClusterNetwork` укажите параметры созданного на предыдущем шаге `ClusterIPAddressPool`.

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: VLAN
  vlan:
    id: 900
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        nic-group: extra
  ipam:
    ipAddressPoolRef:
      kind: ClusterIPAddressPool
      name: public-net-pool

```

После выделения пула IP-адресов для кластерной сети их можно назначать на сетевые интерфейсы подов, подключаемых к этой сети. Подробнее — в разделе «Назначение IP-адресов на сетевые интерфейсы подов, подключаемых к дополнительной сети».

Настройка и подключение underlay-сетей для проброса аппаратных устройств

Эта функция предназначена для высокопроизводительных рабочих нагрузок, требующих прямого доступа к оборудованию, таких как приложения DPDK.

Основные возможности

В DKP реализованы следующие возможности по работе с Underlay-сетями:

- **Проброс аппаратных устройств** — физические сетевые интерфейсы (PF/VF) напрямую предоставляются подам, обходя сетевой стек ядра для максимальной производительности.
- **Настройка SR-IOV** — автоматическая настройка SR-IOV на выбранных Physical Functions для создания Virtual Functions, что позволяет нескольким подам совместно использовать одно и то же оборудование.
- **Поддержка DPDK** — устройства могут быть привязаны в различных режимах, подходящих для рабочих нагрузок DPDK.
 - **VFIO-PCI** — явно подключает сетевое устройство в под, привязывая его к драйверу `vfio-pci`. Внутри пода монтируются соответствующие VFIO dev-устройства (например, `/dev/vfio/vfio0`) для доступа из пользовательского пространства.
 - **DPDK** — универсальный режим, который автоматически выбирает подходящий драйвер для вендора сетевого адаптера. Для сетевых карт Mellanox устройство привязывается к драйверу `mlx5_core` с пробрасыванием как `netdev`-интерфейса, так и необходимых dev-устройств (файлы InfiniBand verbs, `/dev/net/tun`, а также соответствующий `sysfs`-каталог). Для остальных вендоров устройство привязывается через VFIO (также, как в режиме VFIO-PCI).
 - **NetDev** — в под пробрасывается только сетевой интерфейс Linux как стандартное сетевое устройство ядра.

Режимы работы

Поддерживаются следующие режимы выделения устройств, определяющие, как физические интерфейсы предоставляются подам:

- **Shared mode** — создает Virtual Functions (VF) из Physical Functions (PF) с использованием SR-IOV, позволяя нескольким подам совместно использовать одно и то же оборудование. Каждый под получает одну или несколько VF.
- **Dedicated mode** — предоставляет каждый Physical Function как эксклюзивное устройство без SR-IOV. Каждый под получает эксклюзивный доступ к полному PF.

Автоматическая группировка интерфейсов

При включенном `autoBonding` контроллер группирует интерфейсы от нескольких совпавших PF в одно DRA-устройство. Интерфейсы пробрасываются в под как отдельные сетевые интерфейсы, позволяя приложениям (например, DPDK) обрабатывать `bonding`/агрегацию на уровне приложения. Обратите внимание — `bonding`-интерфейсы на уровне ядра внутри пода не

создаются.

Порядок настройки и подключения физических интерфейсов в прикладные поды

Для создания Underlay-сетей для проброса аппаратных устройств в поды используется кастомный ресурс UnderlayNetwork. Он обеспечивает прямое подключение физических сетевых интерфейсов (Physical Functions и Virtual Functions) к подам через Kubernetes Dynamic Resource Allocation (DRA).

Предварительные требования для DPDK-приложений

Перед созданием и настройкой ресурсов UnderlayNetwork необходимо подготовить рабочие узлы кластера для DPDK-приложений:

- настроить hugepages;
- настроить Topology Manager.

Настройка hugepages

DPDK-приложения требуют hugepages для эффективного управления памятью. Настройте hugepages на всех рабочих узлах с помощью NodeGroupConfiguration:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: hugepages-for-dpdk
spec:
  bundles:
    - "*" # Применить ко всем bundle'ам ОС.
  nodeGroups:
    - "*" # Применить ко всем группам узлов.
  weight: 100
  content: |
    #!/bin/bash
    echo "vm.nr_hugepages = 4096" > /etc/sysctl.d/99-hugepages.conf
    sysctl -p /etc/sysctl.d/99-hugepages.conf
```

Эта конфигурация устанавливает `vm.nr_hugepages = 4096` на всех узлах, предоставляя 8 GiB hugepages (4096 страниц × 2 MiB на страницу).

Настройка Topology Manager

Включите Topology Manager на NodeGroup рабочих узлов, где будут запускаться DPDK-приложения. Это обеспечит выделение ресурсов CPU, памяти и устройств из одного NUMA узла.

Пример конфигурации NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  kubelet:
    topologyManager:
      enabled: true
      policy: SingleNumaNode
      scope: Container
  nodeType: Static
```

Для получения дополнительной информации см.:

- topologyManager.enabled;
- topologyManager.policy.

Предварительные действия перед созданием UnderlayNetwork

Перед созданием UnderlayNetwork убедитесь, что:

1. Физические сетевые интерфейсы (NIC) доступны на узлах и обнаружены как ресурсы `NodeNetworkInterface`.
2. Интерфейсы, которые вы планируете использовать, являются Physical Functions (PF), а не Virtual Functions (VF).
3. Сетевые карты поддерживают SR-IOV (для режима Shared).

Проверка и настройка сетевых интерфейсов (ресурсов `NodeNetworkInterface`)

Сначала проверьте, какие Physical Functions доступны на ваших узлах:

```
d8 k get nni -l network.deckhouse.io/nic-pci-type=PF
```

Пример вывода:

NAME	STATE	VF/PF	Binding	Driver	MANAGEDBY	NODE	TYPE	IFNAME	IFINDEX
worker-01-nic-0000:17:00.0	Up	PF	NetDev	ixgbe	Deckhouse	worker-01	NIC	ens3f0	3
worker-01-nic-0000:17:00.1	Up	PF	NetDev	ixgbe	Deckhouse	worker-01	NIC	ens3f1	4
worker-02-nic-0000:17:00.0	Up	PF	NetDev	ixgbe	Deckhouse	worker-02	NIC	ens3f0	3
worker-02-nic-0000:17:00.1	Up	PF	NetDev	ixgbe	Deckhouse	worker-02	NIC	ens3f1	4

Пометьте интерфейсы, которые будут использоваться для UnderlayNetwork:

```
d8 k label nni worker-01-nic-0000:17:00.0 nic-group=dppk
d8 k label nni worker-01-nic-0000:17:00.1 nic-group=dppk
d8 k label nni worker-02-nic-0000:17:00.0 nic-group=dppk
d8 k label nni worker-02-nic-0000:17:00.1 nic-group=dppk
```

i

Вы можете проверить PCI-информацию и статус поддержки SR-IOV для каждого интерфейса с помощью команды:

```
d8 k get nni worker-01-nic-0000:17:00.0 -o json | jq '.status.nic.pci.pf'
```

В секции `status.nic.pci.pf.sriov.supported` можно найти информацию и поддержке SR-IOV.

Создание Underlay-сети в режиме Dedicated

В режиме Dedicated каждый Physical Function предоставляется как эксклюзивное устройство. Этот режим подходит, когда:

- SR-IOV недоступен или не нужен;
- каждому поду требуется эксклюзивный доступ к полному PF.

Чтобы создать Underlay-сеть в режиме Dedicated, выполните следующие шаги:

1. Создайте и примените ресурс UnderlayNetwork. В поле `spec.mode` укажите значение `Dedicated`.

Пример конфигурации:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-dedicated-network
spec:
  mode: Dedicated
  autoBonding: false
  memberNodeNetworkInterfaces:
    - labelSelector:
        matchLabels:
          nic-group: dpdk # Лейбл, которым помечены интерфейсы на этапе проверки и
настройки сетевых интерфейсов.

```

Если `autoBonding` установлен в `true`, все совпавшие PF на узле группируются в одно DRA-устройство, предоставляя поду все PF как отдельные интерфейсы. Когда `false`, — каждый PF публикуется как отдельное DRA-устройство.

2. Проверьте статус созданного UnderlayNetwork:

```
d8 k get underlaynetwork dpdk-dedicated-network -O yaml
```

Пример статуса UnderlayNetwork в режиме Dedicated:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-dedicated-network
...
status:
  observedGeneration: 1
  conditions:
    - message: All 2 member node network interface selectors have matches
      observedGeneration: 1
      reason: AllInterfacesAvailable
      status: "True"
      type: InterfacesAvailable

```

Создание Underlay-сети в режиме Shared

В режиме Shared создаются Virtual Functions (VF) из Physical Functions (PF) с использованием SR-IOV, позволяя нескольким подам совместно использовать одно и то же оборудование. Этот режим требует поддержки SR-IOV на сетевых картах.

Чтобы создать Underlay-сеть в режиме Shared, выполните следующие шаги:

1. Создайте и примените ресурс UnderlayNetwork. В поле `spec.mode` укажите значение `Shared`.

Пример конфигурации:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
spec:
  mode: Shared
  autoBonding: true
  memberNodeNetworkInterfaces:
    - labelSelector:
        matchLabels:
          nic-group: dpdk # Лейбл, которым помечены интерфейсы на этапе проверки и
настройки сетевых интерфейсов.
  shared:
    sriov:
      enabled: true
      numVFs: 8

```

В этом примере:

- `mode: Shared` включает SR-IOV и создание VF;
- `autoBonding: true` группирует одну VF от каждого совпавшего PF в одно DRA-устройство;
- `shared.sriov.enabled: true` включает SR-IOV на выбранных PF;
- `shared.sriov.numVFs: 8` создает 8 Virtual Functions на каждый Physical Function.

i Поля `mode` и `autoBonding` неизменяемы после установки. Тщательно спланируйте конфигурацию перед созданием ресурса.

2. После создания UnderlayNetwork отслеживайте статус конфигурации SR-IOV:

```
d8 k get underlaynetwork dpdk-shared-network -o yaml
```

Пример статуса UnderlayNetwork в режиме Shared:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
...
status:
  observedGeneration: 1
  sriov:
    supportedNICs: 4
    enabledNICs: 4
  conditions:
    - lastTransitionTime: "2025-01-15T10:30:00Z"
      observedGeneration: 1
      message: SR-IOV configured on 4 NICs
      reason: SRIOVConfigured
      status: "True"
      type: SRIOVConfigured
    - lastTransitionTime: "2025-01-15T10:30:05Z"
      observedGeneration: 1
      message: Interfaces are available for allocation
      reason: InterfacesAvailable
      status: "True"
      type: InterfacesAvailable

```

3. Убедитесь, что VF были созданы, проверив ресурс NodeNetworkInterface:


```
d8 k get nni -l network.deckhouse.io/nic-pci-type=VF
```

Подготовка неймспейса для использования UnderlayNetwork

Перед тем как пользователи смогут запрашивать устройства UnderlayNetwork в своих подах, неймспейс должен быть помечен для включения поддержки UnderlayNetwork. Это административная задача, которая должна быть выполнена для неймспейса, где будут запускаться DPDK-приложения.

Чтобы пометить неймспейс для включения поддержки UnderlayNetwork, используйте команду:

```
d8 k label namespace mydpdk direct-nic-access.network.deckhouse.io/enabled=""
```

Настройка и подключение системных сетей (сервисных сетей)

Системные сети (сервисные сети) предназначены для служебного трафика на уровне узлов (например, для нужд хранилища, управления и т. д.) и не используются как дополнительные сети подов.

Дополнительные сервисные сети создаются на узлах кластера поверх существующих underlay-сетей. Для этого используется кастомный ресурс SystemNetwork. Системные сети получают IP-адреса из ClusterIPAddressPool при настройке IPAM.

Принципы и особенности работы системных сетей:

- **Работа поверх underlay-сетей.** Системная сеть подключается к underlay-сети (UnderlayNetwork). Для подключения в параметре `spec.underlayNetworkName` ресурса SystemNetwork указывается имя underlay-сети, к которой должна быть подключена системная сеть. Набор интерфейсов узла (PF или VF), которые будет использовать системная сеть, определяется в параметре `memberNodeNetworkInterfaces` объекта UnderlayNetwork.
- **Поддержка разных типов подключения системной сети к underlay-сети.** Можно создавать VLAN-интерфейсы для подов (`type: VLAN`), использовать прямой доступ к интерфейсам на узлах, подключенным к underlay-сети (`type: Access`) или подключиться через SR-IOV виртуальную функцию (`type: SRIOVirtualFunction`) с опциональной настройкой (MTU, MAC, spoof checking и т. д.).
- **Поддержка механизма IPAM.** Опциональный параметр `spec.ipam` ссылается на ClusterIPAddressPool. Контроллер и агент выделяют адреса из этого пула и назначают их сетевым интерфейсам узла для данной системной сети.
- **Отслеживание статусов системных сетей.** Агент отчитывается об адресах узла (включая IP-адреса системных сетей) в ресурсах NodeNetworkStatus. Внутренние ресурсы SystemNetworkNodeNetworkInterfaceAttachment отслеживают привязку каждой системной сети к родительскому интерфейсу на каждом узле.

Предварительные требования для создания и использования системных сетей

Для создания и использования системных сетей в кластере необходимо выполнение следующих требований:

1. Должна существовать underlay-сеть. Системная сеть подключается к ней с помощью параметра `spec.underlayNetworkName`. Селекторы, указанные в параметре `memberNodeNetworkInterfaces` объекта UnderlayNetwork, определяют, какие интерфейсы узлов используются системной сетью.
2. **Опционально.** Для автоматической выдачи IP-адресов на интерфейсах, принадлежащих системной сети, создайте пул адресов для системной сети. Его необходимо указать в параметре `spec.ipam.clusterIPAddressPoolName` ресурса SystemNetwork при создании сети.

Создание системной сети

Для создания системной сети используйте ресурс SystemNetwork. Поддерживается создание системных сетей со следующими типами подключения к underlay-сетям, поверх которых они будут работать:

- `VLAN` — на интерфейсах, подключенных к underlay-сети (underlay-интерфейсах) создаются VLAN-интерфейсы;
- `Access` — прямой доступ (без VLAN) к интерфейсам на узлах, подключенным к underlay-сети;
- `SRIOVirtualFunction` — подключение к физическому интерфейсу через SR-IOV.

Тип VLAN

Создаёт VLAN-интерфейс на каждом совпавшем underlay-интерфейсе. При создании сети требуется указать VLAN ID (параметр `spec.vlan.id`).

Пример манифеста системной сети с типом подключения к underlay-сети `VLAN`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: storage-network
spec:
  type: VLAN
  underlayNetworkName: my-underlay
  vlan:
    id: 100
  ipam:
    clusterIPAddressPoolName: storage-pool
```

Для проверки статуса сети после создания воспользуйтесь разделом «Проверка статуса системной сети».

Тип Access

Использует underlay-интерфейс напрямую (без VLAN). Удобно, когда underlay уже представляет один L2-сегмент.

Пример манифеста системной сети с типом подключения к underlay-сети `Access`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: mgmt-network
spec:
  type: Access
  underlayNetworkName: my-underlay
  ipam:
    clusterIPAddressPoolName: mgmt-pool
```

Для проверки статуса сети после создания воспользуйтесь разделом «Проверка статуса системной сети».

Тип SRIOVirtualFunction

Подключение через SR-IOV виртуальную функцию. Физический сетевой интерфейс должен быть в режиме `Shared`, чтобы существовали VF. Опционально можно настроить параметры VF.

Пример манифеста системной сети с типом подключения к underlay-сети `SRIOVirtualFunction`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: vf-service-network
spec:
  type: SRIOVirtualFunction
  underlayNetworkName: dpdk-shared-network
  sriovVirtualFunction:
    vlan: 200
    mtu: 1500
    linkState: Auto
  ipam:
    clusterIPAddressPoolName: vf-pool
```

Для проверки статуса сети после создания воспользуйтесь разделом «Проверка статуса

системной сети».

Создание пула IP-адресов для настройки IPAM системной сети

Чтобы создать пул адресов для настройки IPAM системной сети, используйте ресурс ClusterIPAddressPool:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterIPAddressPool
metadata:
  name: storage-pool
spec:
  leaseTTL: 24h
  pools:
    - network: 10.20.30.0/24
      ranges:
        - 10.20.30.10-10.20.30.250
```

При создании системной сети укажите этот пул в параметре `spec.ipam.clusterIPAddressPoolName` ресурса `SystemNetwork`.

Проверка статуса системной сети

Для получения списка системных сетей используйте команду:

```
d8 k get systemnetworks
```

Для просмотра статуса конкретной системной сети используйте команду:

```
d8 k get systemnetwork storage-network -o yaml
```

В `status` отображаются:

- `nodeAttachmentsCount` — общее число привязок (по одной на совпавший интерфейс узла);
- `readyNodeAttachmentsCount` — привязки в состоянии готовности (настроены и работают);
- `conditions` — например, `Ready`, когда все привязки готовы.

Для просмотра внутренних привязок (по одной на пару «системная сеть + родительский интерфейс») используйте команду:

```
d8 k get systemnetworknodenetworkinterfaceattachments
```

Для просмотра IP-адресов на уровне узла (включая IP системных сетей) для всех узлов используйте команду (на каждый узел назначается по одному `NodeNetworkStatus`):

```
d8 k get nodenetworkstatus
```

Для просмотра информации об IP-адресах на уровне узла (включая IP системных сетей) для конкретного узла используйте команду:

```
d8 k get nodenetworkstatus -l network.deckhouse.io/node-name=worker-01 -o yaml
```

В `status.addresses` ищите записи с `type: SystemNetworkIP` и полем `systemNetworkName`, значение которого равно имени вашей системной сети.

Дополнительные ресурсы

- Кастомные ресурсы модуля `node-manager`
- Документация модуля `sdn`

Управление статичными маршрутами и правилами ip rule на узлах кластера

В Deckhouse Kubernetes Platform управление статичными маршрутами и правилами ip rule на

узлах кластера реализуется с помощью модуля `static-routing-manager`.

Примеры настроек

Создание маршрута в основной таблице «main»

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-main
spec:
  ipRoutingTableID: 254 # Идентификатор основной таблицы маршрутизации – 254.
  routes:
    - destination: 10.0.0.0/8
      gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
```

На узлах, попадающих под `nodeSelector`, будет создан маршрут `10.0.0.0/8 via 192.168.0.1`:

```
ip -4 route ls
```

Пример вывода:

```
10.0.0.0/8 via 192.168.0.1 dev eth0 realm 216
```

Инструкция `realm 216` в маршруте используется как маркер для идентификации маршрута под управлением модуля (`d8 hex = 216 dec`).

Создание маршрута в дополнительной таблице

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-extra
spec:
  routes:
    - destination: 0.0.0.0/0
      gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
status:
  ipRoutingTableID: 10000 # Если spec.ipRoutingTableID не указан, он будет
    сгенерирован автоматически и размещён в status.
```

На узлах, попадающих под `nodeSelector`, будет создан маршрут `0.0.0.0/0 via 192.168.0.1` в таблице `10000`:

```
ip -4 route ls table 10000
```

Пример вывода:

```
default via 192.168.0.1 dev eth0 realm 216
```

Создание ip rule

```

apiVersion: network.deckhouse.io/v1alpha1
kind: IPRuleSet
metadata:
  name: myiprule
spec:
  rules:
  - selectors:
    from:
    - 192.168.111.0/24
    - 192.168.222.0/24
    to:
    - 8.8.8.8/32
    - 172.16.8.0/21
    sportRange:
      start: 100
      end: 200
    dportRange:
      start: 300
      end: 400
    ipProto: 6
    actions:
      lookup:
        routingTableName: myrt-extra
      priority: 50
  nodeSelector:
    node-role.deckhouse.io: load-balancer

```

На узлах, попадающих под `nodeSelector`, будет создан ip rule. Команда для просмотра:

```
ip rule list
```

Пример вывода:

```

...
50: from 192.168.111.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.222.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216
50: from 192.168.222.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.111.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216

```

Дополнительные ресурсы

- Документация модуля `static-routing-manager`

Управление DNS в кластере Kubernetes

Управление DNS в кластере Kubernetes реализуется с помощью модуля `kube-dns`.

Модуль устанавливает компоненты CoreDNS для управления DNS в кластере Kubernetes.

i Модуль удаляет объекты Deployment, ConfigMap и RBAC для CoreDNS, установленные через утилиту `kubeadm`. При разворачивании собственного CoreDNS избегайте использования имен `coredns` или `system:coredns` для любых ресурсов (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Используйте альтернативные имена, например `infra-dns`, чтобы предотвратить их автоматическое удаление Deckhouse Kubernetes Platform.

Пример конфигурации

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
      - 8.8.8.8
      - 8.8.4.4
    hosts:
      - domain: one.example.com
        ip: 192.168.0.1
      - domain: two.another.example.com
        ip: 10.10.0.128
    stubZones:
      - zone: consul.local
        upstreamNameservers:
          - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
      - foo.bar
      - baz.qux

```

Дополнительные ресурсы

- Документация модуля `kube-dns`

Кеширование DNS-запросов на узлах кластера

В Deckhouse Kubernetes Platform можно развернуть локальный кеширующий DNS-сервер на каждом узле кластера. Он экспортирует метрики в Prometheus для визуализации в дашборде Grafana.

Функциональность реализуется модулем `node-local-dns`. Модуль состоит из оригинального CoreDNS, разворачиваемого в DaemonSet на всех узлах кластера, с добавлением алгоритма настройки сети и правил iptables.

Подробная информация о проблемах, которые позволяет решить кеширующий DNS-сервер и о принципе его работы — в разделе кеширующий DNS-сервер в кластере.

Пример настройки кастомного DNS в поде

```

apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx

```

Подробную информацию о настройке DNS можно найти в документации Kubernetes.

Дополнительные ресурсы

- Документация модуля `node-local-dns`

Режимы работы bpfLB для обработки внешнего трафика

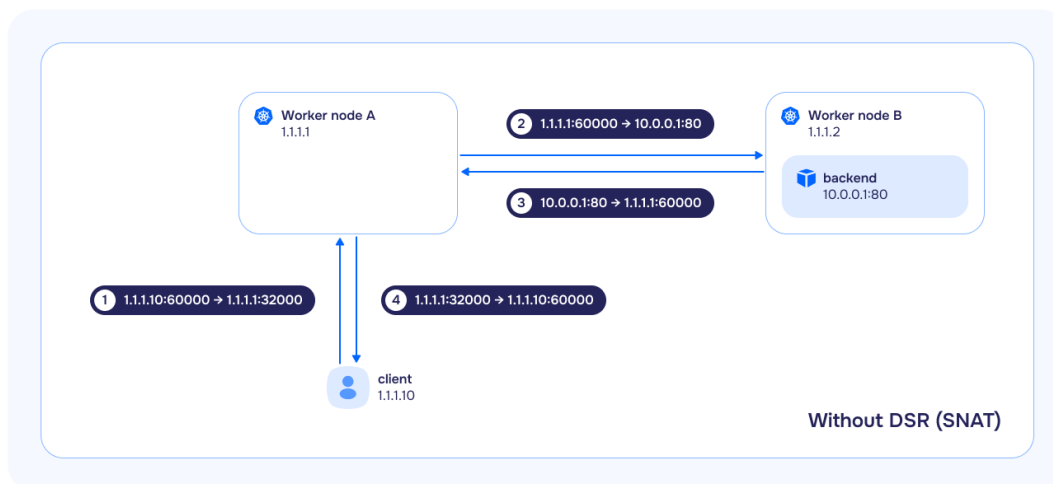
Для настройки режима работы BPF Load Balancer (bpfLB) в Deckhouse Kubernetes Platform можно использовать модуль `cni-cilium`.

В Kubernetes трафик обычно проходит через балансировщик, который распределяет входящие запросы по серверам и обрабатывает обратные ответы. Это ограничивает общую пропускную способность ресурсами балансировщика.

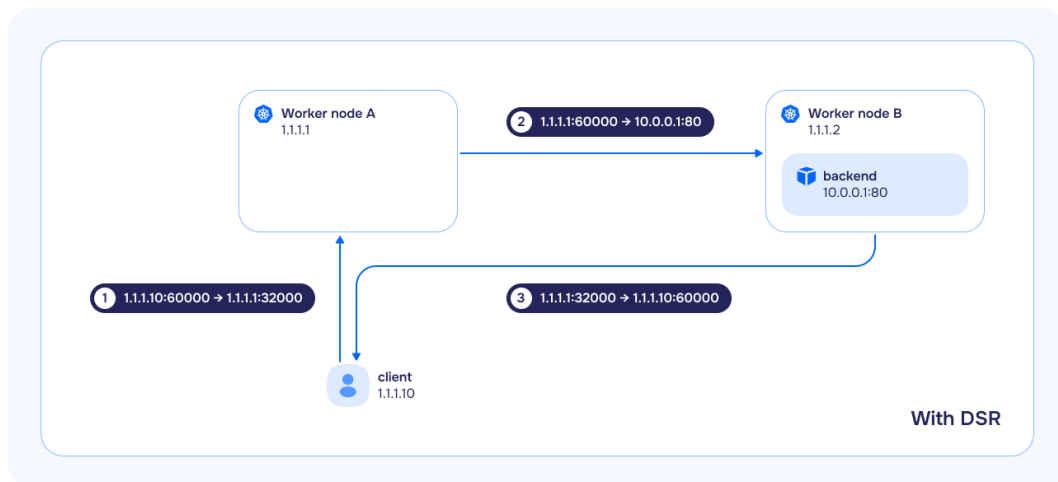
Для оптимизации трафика используется механизм DSR, в котором входящие пакеты проходят через балансировщик, а исходящие идут напрямую с терминирующих серверов. Так как обычно ответы имеют много больший размер чем запросы, то такой подход позволяет значительно увеличить общую пропускную способность схемы.

В модуле возможен выбор режима работы с помощью параметра `bpfLBMode`, влияющего на поведение сервисов с типом `NodePort` и `LoadBalancer`:

- **SNAT** (Source Network Address Translation) — один из подвидов NAT, при котором для каждого исходящего пакета происходит трансляция IP-адреса источника в IP-адрес шлюза из целевой подсети, а входящие пакеты, проходящие через шлюз, транслируются обратно на основе таблицы трансляций. В этом режиме bpfLB полностью повторяет логику работы kube-proxy:
 - если для объекта Service указано `externalTrafficPolicy: Local`, то трафик будет передаваться и балансироваться только в те целевые поды, которые запущены на том же узле, на который этот трафик пришел. Если целевой под не запущен на этом узле, то трафик будет отброшен.
 - если для объекта Service указано `externalTrafficPolicy: Cluster`, то трафик будет передаваться и балансироваться во все целевые поды в кластере. При этом, если целевые поды находятся на других узлах, то при передаче трафика на них будет произведен SNAT (IP-адрес источника будет заменен на InternalIP узла).



- **DSR** - (Direct Server Return) — метод, при котором весь входящий трафик проходит через балансировщик нагрузки, а весь исходящий трафик обходит его. Такой метод используется вместо SNAT. Часто ответы имеют много больший размер чем запросы и DSR позволяет значительно увеличить общую пропускную способность схемы:
 - если в Service указан `externalTrafficPolicy: Local`, то поведение абсолютно аналогично kube-proxy и bpfLB в режиме SNAT.
 - если в Service указан `externalTrafficPolicy: Cluster`, то трафик так же будет передаваться и балансироваться во все целевые поды в кластере. При этом важно учитывать следующие особенности:
 - если целевые поды находятся на других узлах, то при передаче на них входящего трафика будет сохранен IP-адрес источника;
 - исходящий трафик пойдет прямо с узла, на котором был запущен целевой под;
 - IP-адрес источника будет заменен на внешний IP-адрес узла, на которую изначально пришел входящий запрос.



i

В случае использования режима DSR и Service с `externalTrafficPolicy: Cluster` требуются дополнительные настройки сетевого окружения. Сетевое оборудование должно быть готово к асимметричному прохождению трафика: отключены или настроены соответствующим образом средства фильтрации IP адресов на входе в сеть (uRPF, sourceGuard и т.п.).

- **Hybrid** — в данном режиме TCP-трафик обрабатывается в режиме DSR, а UDP — в режиме SNAT.

Пример настройки режима работы bpflB:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  settings:
    tunnelMode: VXLAN
    bpflBMode: SNAT # Выбран режим SNAT.
  version: 1
  enabled: true
```

Дополнительные ресурсы

- Документация модуля `cni-cilium`

Синхронизация времени на узлах

Для синхронизации времени на узлах кластера Deckhouse используется встроенное решение на основе `chrony`. В решении используется протокол Network Time Protocol (NTP), который обеспечивает синхронизацию системных часов на узлах кластера с внешними NTP-серверами. При необходимости вы можете отключить встроенный механизм и использовать собственные NTP-демоны.

Включение встроенной синхронизации времени

Включите модуль `chrony`, чтобы включить синхронизацию времени:

```
d8 system module enable chrony
```

По умолчанию в качестве источника времени используется сервер `pool.ntp.org`. Указать список NTP-серверов можно с помощью параметра `ntpServers` конфигурации модуля `chrony`.

Пример конфигурации модуля с указанием NTP-серверов:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - ntp.ubuntu.com
      - time.google.com
version: 1

```

Использование собственных NTP-демонов

Чтобы отключить встроенный механизм синхронизации времени и использовать собственные NTP-демоны на узлах, выполните следующие шаги:

1. Отключите модуль `chrony`:

```
d8 system module disable chrony
```

При успешном выполнении команды будет выведено сообщение о том, что модуль был отключён:

```
Module chrony disabled
```

2. Создайте ресурс `NodeGroupConfiguration`, чтобы включить NTP-демоны на узлах.

Пример для `systemd-timesyncd`:

```

d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: enable-ntp-on-node.sh
spec:
  weight: 100
  nodeGroups: ["*"]
  bundles: ["*"]
  content: |
    systemctl enable systemd-timesyncd
    systemctl start systemd-timesyncd
EOF

```

Дополнительные ресурсы

- Документация модуля `chrony`
- Кастомные ресурсы модуля `node-manager`

Доступ через OpenVPN

В разделе описывается настройка доступа к ресурсам кластера через OpenVPN с аутентификацией по сертификатам.

Настройки доступны в простом веб-интерфейсе, который позволяет:

- Выпускать сертификаты;
- Отзывать сертификаты;
- Отменять отзыв сертификатов;
- Получать готовый пользовательский конфигурационный файл.

OpenVPN интегрируется с аутентификацией пользователей, что позволяет управлять доступом в веб-интерфейс.

Настройка

Доступ к OpenVPN снаружи кластера

Настроить публикацию OpenVPN-сервера можно несколькими способами:

1. Выберите для подключения один или несколько внешних IP-адресов.
2. Воспользуйтесь одним из методов подключения:
 - по внешнему IP-адресу (`ExternalIP`) — если есть узлы с публичными IP-адресами;
 - с помощью `LoadBalancer` — для всех облачных провайдеров и их схем размещения с поддержкой заказа `LoadBalancer`;
 - `Direct` — настройте путь трафика вручную: от точки входа в кластер до пода с OpenVPN.

Пример для bare-metal кластеров

Примените в кластере следующий YAML-файл:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: ExternalIP
    externalIP: 5.4.54.4 # Внешний IP-адрес.
```

Пример для AWS и Google Cloud

Примените в кластере следующий YAML-файл:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

Пример для публичного IP-адреса на внешнем балансировщике

Примените в кластере следующий YAML-файл:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # Внутренний IP-адрес, который примет трафик от внешнего
    балансировщика.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node
```

Аутентификация

Для доступа в OpenVPN возможно настроить аутентификацию пользователей. Также можно настроить аутентификацию с помощью параметра `externalAuthentication`. Если эти варианты отключены, будет включена базовая аутентификация со сгенерированным паролем.

Чтобы просмотреть сгенерированный пароль, выполните команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values openvpn -o json | jq '.openvpn.internal.auth.password'
```

Чтобы сгенерировать новый пароль, удалите секрет:

```
d8 k -n d8-openvpn delete secret/basic-auth
```



Параметр `auth.password` больше не поддерживается.

Доступные ресурсы кластера после подключения к VPN

На компьютер пользователя после подключения к VPN доставляются (push) следующие параметры:

- Адрес `kube-dns` добавляется в DNS-серверы клиента для возможности прямого обращения к сервисам Kubernetes по FQDN;
- Маршрут в локальную сеть;
- Маршрут в сервисную сеть кластера;
- Маршрут в сеть подов.

Аудит пользовательских соединений

Для мониторинга пользовательских соединений возможно включить логирование пользовательской активности через VPN в JSON-формате. Группировка трафика происходит по полям `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto`. С помощью `log-shipper` логи контейнера можно собрать и отправить на хранение для последующего аудита.

Почему не работает автоматическая настройка DNS-сервера при подключении на macOS и Linux с помощью клиента OpenVPN

В связи с архитектурными особенностями операционных систем семейства Linux и macOS автоматическая конфигурация DNS-сервера при подключении с помощью официального клиента OpenVPN невозможна.

Для настройки DNS-сервера в таких ОС сервисом предусмотрена возможность использования сторонних скриптов, которые запускаются при подключении и отключении клиента.

В клиентских конфигурациях, генерируемых модулем, предопределены и закомментированы блоки, отвечающие за эти настройки:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

Для активации указанных блоков кода необходимо их раскомментировать (удалить начальный символ `#`), а также указать корректные пути к скриптам.

Скрипты можно подготовить самостоятельно, или воспользоваться готовыми решениями от официального OpenVPN Community (для Linux). Для macOS можно воспользоваться сторонним скриптом.



Скрипты должны обладать правами на исполнение.

Как отозвать, ротировать или удалить сертификат клиента

Все действия с клиентскими сертификатами выполняются через веб-интерфейс `openvpn-admin`. Справа от имени каждого пользователя доступны кнопки для управления сертификатом:

Search Table						Add user	Hide revoked
	Name ↕	Account Status ↕	Active Connections ↕	Expiration Date ↕	Revocation Date ↕	Actions	
1	test	Active	0	2035-06-15 15:28:14		Revoke	Download config Edit routes

Чтобы ротировать (выпустить новый сертификат) или удалить клиента, необходимо сначала отозвать его текущий сертификат (Revoke):

Search Table						Add user	Hide revoked
	Name ↕	Account Status ↕	Active Connections ↕	Expiration Date ↕	Revocation Date ↕	Actions	
1	test	Revoked	0	2035-06-15 15:28:14	2025-06-18 08:57:51	Delete Rotate	Unrevoke

После отзыва становятся доступными действия Renew (ротация) и Delete (удаление).

Как ротировать сертификат сервера

Серверный сертификат ротится автоматически за 1 день до окончания срока его действия.

Если требуется выполнить ротацию вручную (например, при повреждении сертификата или внеплановой замене), выполните следующие шаги:

1. Удалите текущий секрет, содержащий сертификат и ключ сервера:

```
d8 k -n d8-openvpn delete secrets openvpn-pki-server
```

2. Перезапустите под OpenVPN, чтобы инициировать генерацию нового сертификата:

```
d8 k -n d8-openvpn rollout restart sts openvpn
```

Как ротировать корневой сертификат (CA)

Корневой сертификат (CA) и серверный сертификат ротится автоматически за 1 день до окончания срока действия. Автоматическая ротация сертификатов пользователя не предусмотрена. Корневой сертификат (CA) используется для подписи всех сертификатов в OpenVPN — как серверных, так и клиентских. Поэтому при его замене необходимо перевыпустить все зависимые сертификаты.

Шаги для ротации корневого сертификата:

1. Отзовите или удалите все активные клиентские сертификаты. Сделать это можно через интерфейс `openvpn-admin`. Если вы воспользуетесь отзывом, то после замены CA можно будет выполнить ротацию сертификатов (Renew), не создавая клиента заново.
2. Удалите секреты `openvpn-pki-ca` и `openvpn-pki-server` в пространстве имён `d8-openvpn`:

```
d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
```

3. Перезапустите поды OpenVPN:

```
d8 k -n d8-openvpn rollout restart sts openvpn
```

4. Выполните ротацию сертификатов для отозванных клиентов или создайте новых клиентов с новыми сертификатами.

5. Удалите все отозванные сертификаты:

```
d8 k -n d8-openvpn delete secrets -l revokedForever=true
```

Дополнительные ресурсы

- Документация модуля `log-shipper`
- Параметры модуля `openvpn`

Хранилища

Надёжное хранение данных — одна из ключевых задач при развёртывании и эксплуатации кластеров Kubernetes. В Deckhouse Kubernetes Platform этот вопрос решается за счёт гибкой поддержки программно-определяемых и внешних хранилищ, а также удобных инструментов автоматизации и управления.

В данном разделе вы узнаете:

- какие типы хранилищ поддерживает Deckhouse;
- как организовать локальные (SDS) и внешние системы хранения;
- какими средствами Deckhouse упрощает настройку и эксплуатацию хранилищ;
- как выбрать оптимальное решение для ваших сценариев, исходя из требований к надёжности, производительности и масштабируемости.

Подробнее с разметкой и рекомендуемыми объемами дисков можно ознакомиться в руководстве «Разметка и объем дисков».

Поддерживаемые типы хранилищ

Deckhouse Kubernetes Platform предлагает широкий спектр решений, которые условно делятся на две большие группы.

Программно-определяемые хранилища

- Управление логическими томами на узлах кластера
- Локальное хранилище Local Path Provisioner
- Локальное хранилище на основе LVM (Logical Volume Manager)
- Реплицируемое хранилище на основе DRBD (Distributed Replicated Block Device)

Внешние хранилища

- Распределённое хранилище Ceph
- Хранилище данных HPE
- Хранилище данных Huawei
- Хранилище данных NFS
- Объектное хранилище на основе S3
- Хранилище данных на основе протокола SCSI
- Унифицированное хранилище TATLIN.UNIFIED (Yadro)

Ключевые возможности

- Все настройки хранения выполняются через Deckhouse и его модули, упрощая интеграцию со всеми компонентами кластера.
- Для Ceph, NFS, S3, корпоративных СХД и других систем предусмотрены готовые решения, которые упрощают подключение хранилища к кластеру.
- За счёт поддержки SDS (в том числе с DRBD) и распределённых решений (Ceph, S3) можно масштабироваться при росте нагрузки.
- Репликация данных и интеграция с мощными системами СХД помогают защитить критичные сервисы и приложения от потери данных.

Настройка хранилищ

Настройка хранилищ происходит в несколько шагов, которые зависят от выбранного типа хранилища. Основные этапы настройки:

- Включение и конфигурирование соответствующих модулей.
- Создание групп томов (Volume Groups).
- Подготовка и создание объектов StorageClass, их последующее назначение и использование.

У каждого типа хранилища могут быть свои специфические требования и нюансы конфигурации, описанные в соответствующих руководствах.

Создание StorageClass

Для создания объектов StorageClass необходимо подключить одно или несколько хранилищ, которые управляют ресурсами PersistentVolume. Созданные объекты StorageClass можно использовать для организации виртуальных дисков и образов. Подробнее о создании и использовании StorageClass можно узнать в соответствующих разделах документации по каждому типу хранилища.

Назначение StorageClass по умолчанию

StorageClass по умолчанию используется в случаях, когда при создании ресурсов PersistentVolumeClaim явно не указан класс хранения. Это упрощает процесс создания и использования хранилищ, избегая необходимости каждый раз указывать класс вручную.

Чтобы задать StorageClass по умолчанию, укажите нужный класс хранения в глобальной конфигурации. Пример команды:

```
# Укажите имя своего объекта StorageClass.
DEFAULT_STORAGE_CLASS=replicated-storage-class
d8 k patch mc global --type='json' -p='[{"op": "replace", "path":
"/spec/settings/defaultClusterStorageClass", "value":
"'"$DEFAULT_STORAGE_CLASS"'"}]'
```

Дополнительные ресурсы

- Глобальные параметры

Настройка создания снимков томов

Deckhouse Kubernetes Platform поддерживает создание снимков томов для CSI-драйверов в кластере Kubernetes.

Снимки фиксируют состояние тома на определенный момент времени и могут быть использованы для восстановления данных или клонирования томов. Способность создавать снимки зависит от возможностей используемого CSI-драйвера.

Поддерживаемые CSI-драйверы

Создание снимков поддерживается следующими CSI-драйверами:

- Облачные ресурсы провайдера OpenStack;
- Облачные ресурсы провайдера VMWare vSphere;
- Распределённое хранилище Ceph;
- Облачные ресурсы провайдера Amazon Web Services;
- Облачные ресурсы провайдера Microsoft Azure;
- Облачные ресурсы провайдера Google Cloud Platform;
- Реплицируемое хранилище на основе DRBD;
- Хранилище данных NFS.

Создание снимков

Перед созданием снимков убедитесь, что в кластере настроены объекты VolumeSnapshotClass. Список доступных классов можно получить командой:

```
d8 k get volumesnapshotclasses.snapshot.storage.k8s.io
```

Чтобы создать снимок для тома, укажите нужный VolumeSnapshotClass в манифесте:

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-snapshot
spec:
  volumeSnapshotClassName: <имя-класса>
  source:
    persistentVolumeClaimName: <имя-PVC>

```

Восстановление из снимка

Чтобы восстановить данные из снимка, создайте PVC, ссылающийся на ранее созданный объект VolumeSnapshot:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restored-pvc
spec:
  dataSource:
    name: example-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  storageClassName: <имя-StorageClass>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi

```



Не все CSI-драйверы поддерживают восстановление тома из снимка. Убедитесь, что используемый драйвер поддерживает соответствующие возможности.

Дополнительные ресурсы

- Документация модуля `cloud-provider-aws`
- Документация модуля `cloud-provider-azure`
- Документация модуля `cloud-provider-gcp`
- Документация модуля `cloud-provider-openstack`
- Документация модуля `cloud-provider-vsphere`

Локальное хранилище Local Path Provisioner

Deckhouse Kubernetes Platform предоставляет возможность настраивать локальные хранилища Local Path Provisioner. Это простое решение без поддержки снимков и ограничений на размер, которое лучше всего подходит для разработки, тестирования и небольших кластеров. Данное хранилище использует локальное дисковое пространство узлов Kubernetes для создания PersistentVolume, не полагаясь на внешние системы хранения данных.

Принцип работы

Для каждого ресурса LocalPathProvisioner создается соответствующий объект StorageClass. Список узлов, на которых разрешено использовать StorageClass, определяется на основе поля `nodeGroups` и используется при размещении подов.

При запросе диска подом происходит следующее:

- создаётся PersistentVolume с типом `HostPath`;
- на нужном узле создается директория, путь к которой формируется из параметра `path`, имени PV и PVC.

Пример пути:

```
/opt/local-path-provisioner/pvc-d9bd3878-f710-417b-a4b3-38811aa8aac1_d8-monitoring_prometheus-main-db-prometheus-main-0
```

Ограничения

- Невозможно задать ограничение на размер создаваемого тома.

Примеры ресурсов LocalPathProvisioner

ReclaimPolicy: Retain (по умолчанию)

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
    - system
  path: "/opt/local-path-provisioner"
```

ReclaimPolicy: Delete

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
    - system
  path: "/opt/local-path-provisioner"
  reclaimPolicy: "Delete"
```

Настройка Prometheus с использованием локального хранилища

1. Примените ресурс LocalPathProvisioner:

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
    - system
  path: "/opt/local-path-provisioner"
```

2. Убедитесь, что `spec.nodeGroups` соответствует NodeGroup, на котором будет работать Prometheus.
3. Укажите имя созданного StorageClass в настройках Prometheus:

```
longtermStorageClass: localpath-system
storageClass: localpath-system
```

4. Дождитесь перезапуска подов Prometheus.

Дополнительные ресурсы

- Кастомные ресурсы модуля `local-path-provisioner`

Настройка узлов для программно-определяемых хранилищ

Deckhouse Kubernetes Platform предоставляет возможность автоматического управления логическими томами (Logical Volume Manager, LVM) на узлах кластера с помощью пользовательских ресурсов Kubernetes. Эта функциональность обеспечивается модулем `sds-node-configurator` и включает в себя:

- Обнаружение блочных устройств на каждом узле и создание соответствующих ресурсов `BlockDevice`.

i Ручное создание и изменение ресурса `BlockDevice` запрещено.

- Автоматический поиск на узлах `LVMVolumeGroup` с лейблом `storage.deckhouse.io/enabled=true` (включая `thin pool`) и управление соответствующими ресурсами `LVMVolumeGroup`. При обнаружении групп томов (Volume Group, VG) без ресурсов контроллер создаёт ресурс `LVMVolumeGroup` автоматически.
- Регулярное сканирование физических томов (Physical Volume, PV) на узлах, входящих в управляемые VG. При расширении нижестоящего блочного устройства контроллер выполняет `pvresize` для соответствующего физического тома и автоматически увеличивает размер всех логических томов, привязанных к этой группе.

i Уменьшение размеров блочного устройства не поддерживается.

- Синхронизация состояния `LVMVolumeGroup` с реальным состоянием узла: создание новых групп томов согласно `LVMVolumeGroup`, расширение существующих при изменении `desiredCapacity` и удаление группы при удалении ресурса. Подробные примеры работы см. в разделе Примеры работы с `LVMVolumeGroup`.

Дополнительные ресурсы

- Документация модуля `sds-node-configurator`

Примеры использования

i Работоспособность гарантируется только при использовании стоковых ядер, поставляемых вместе с поддерживаемыми дистрибутивами. При использовании нестандартных ядер или дистрибутивов поведение может быть непредсказуемым.

Контроллер оперирует двумя типами кастомных ресурсов Kubernetes:

- `BlockDevice` — ресурс, представляющий блочное устройство;
- `LVMVolumeGroup` — ресурс, описывающий группу томов Logical Volume Manager.

Работа с ресурсами `BlockDevice`

Создание ресурса `BlockDevice`

Контроллер периодически сканирует доступные блочные устройства на каждом узле и автоматически формирует ресурс `BlockDevice` при обнаружении устройства, соответствующего правилам. В результате создаётся объект ресурса с уникальным именем, содержащий детальную информацию о характеристиках устройства.

Критерии отбора устройства контроллером

- Устройство не используется в качестве DRBD-тома (Distributed Replicated Block Device);
- устройство не является loop-интерфейсом (виртуальным блочным устройством);
- устройство является логическим томом (Logical Volume);
- файловая система отсутствует или имеет маркировку `LVM2_MEMBER` (указывает на принадлежность устройству LVM);
- устройство не содержит разделов (отсутствует `partition table`);
- ёмкость устройства превышает 1 GiB;
- для виртуальных дисков обязательно наличие серийного номера.

Сформированный ресурс `BlockDevice` служит источником данных для последующей работы с ресурсами `LVMVolumeGroup`.

Обновление ресурса `BlockDevice`

При изменении состояния блочного устройства (например, изменение размера, изменение метаданных или исчезновение устройства из системы) контроллер автоматически обнаруживает эти изменения и обновляет поля ресурса `BlockDevice`, обеспечивая актуальность информации. Пользовательская правка ресурса `BlockDevice` запрещена и будет перезаписана контроллером.

Удаление ресурса `BlockDevice`

Контроллер удаляет ресурс `BlockDevice` только при выполнении условий:

- Ресурс находится в состоянии `Consumable` (доступен для потребления в Logical Volume Manager).
- Блочное устройство более не доступно в системе (устройство было удалено или отключено).
- Блочное устройство не входит в состав группы томов с лейблом `storage.deckhouse.io/enabled=true` (такая группа томов управляется `LVMVolumeGroup` и удаление `BlockDevice` не производится).

i Удаление ресурса `BlockDevice` происходит без участия пользователя. При ручном удалении ресурса контроллер повторно создаст его при следующем сканировании, если устройство по-прежнему удовлетворяет критериям.

Работа с ресурсами `LVMVolumeGroup`

Ресурс `LVMVolumeGroup` объединяет несколько ресурсов `BlockDevice` в единую группу томов Logical Volume Manager (LVM) на узле и отображает текущее состояние этой группы.

i На текущий момент поддерживается только тип `Local` (локальные группы томов).

Создание ресурса `LVMVolumeGroup`

Существует два сценария формирования ресурса `LVMVolumeGroup`:

- Автоматическое создание ресурса:
 - Контроллер сканирует список активных групп томов LVM на каждом узле.
 - Если у обнаруженной группы томов присутствует лейбл `storage.deckhouse.io/enabled=true` и нет соответствующего ресурса `LVMVolumeGroup`, контроллер автоматически создаёт ресурс.
 - При этом контроллер заполняет секцию `spec`, за исключением поля `thinPools`. Все остальные параметры (имена блочных устройств, имя Volume Group и т.д.) подтягиваются автоматически из состояния системы.
 - Для управления пулами малого размера (`thin pool`) пользователь может вручную добавить информацию о них в секцию `spec` после создания ресурса.
- Ручное создание ресурса:
 - Пользователь формирует YAML-манифест, указывая минимальный набор полей:
 - `metadata.name` — уникальное имя ресурса `LVMVolumeGroup`;
 - `spec` — имя узла, на котором будет существовать группа томов.
 - После валидации контроллер создаёт группу томов на узле и добавляет в кастомный ресурс актуальную информацию о состоянии созданной LVM Volume Group.

Пример создания локальной Volume Group без `thin pool`:

```

apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd6441267
  actualVGNameOnTheNode: "vg-0"

```

```

apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"

```

Пример создания локальной Volume Group с thin pool (250 GiB):

```

apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd6441267
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi

```

```

apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi

```

Пользователь может использовать любые допустимые селекторы (`matchLabels` или `matchExpressions`) для выбора `BlockDevice`:

- `matchLabels` позволяет выбрать все устройства, у которых совпадает указанный лейбл (например, `kubernetes.io/hostname=node-0`);
- `matchExpressions` позволяет задать более гибкие выражения, например, включение перечисленных имён устройств.

Поле `spec.local.nodeName` должно совпадать с именем узла, на котором создаётся `Volume Group`. В противном случае ресурс не будет запущен. Все выбираемые блочные устройства должны физически находиться на одном узле для ресурса с типом `Local`.



После применения манифеста контроллер автоматически создаст или обновит инфраструктуру LVM на узле, включая работу с `physical volumes` (`pvccreate`), созданием или изменением группы томов (`vgcreate/vgextend`) и настройкой `thin pool` (`lvcreate`).

Обновление ресурса LVMVolumeGroup

Для изменения конфигурации группы томов (добавление/удаление блочных устройств или изменение параметров `thin pool`) достаточно отредактировать секцию `spec` ресурса. Контроллер выполнит валидацию новых параметров и скорректирует состояние на узле (выполнит команды `vgextend`, `lvcreate`, `lvresize` и т.д.).

Секция `status` ресурса автоматически отражает актуальную информацию о состоянии `Volume Group` (список `physical volumes`, данные о `thin pool`, свободное/занятое пространство и т.д.).



- Контроллер не изменяет `spec` автоматически — он только читает желаемое состояние и приводит фактическое состояние узла в соответствие. Все изменения желаемого состояния должны вноситься пользователем в `spec`.
- Не рекомендуется вносить правки в `status`, так как этот раздел управляется контроллером.

Удаление ресурса LVMVolumeGroup

Контроллер удаляет ресурс автоматически, если на узле больше не существует `Volume Group` (например, все блочные устройства отключены или удалены). Для ручного удаления `Volume Group` и связанных с ней логических томов пользователь может вызвать следующую команду:

```
d8 k delete lvg <имя-ресурса>
```

После этого контроллер выполнит полномасштабное удаление физических и логических объектов LVM (`lvremove`, `vgremove` и т. д.).



Если в Volume Group присутствуют логические тома (включая thin pool), пользователь должен сначала удалить или переместить их (команды `lvremove` или `pvmove + vgreduce`). В противном случае контроллер не сможет удалить ресурс. Чтобы предотвратить случайное удаление ресурса, можно назначить аннотацию `storage.deckhouse.io/deletion-protection` — пока она присутствует, контроллер не будет инициировать удаление группы томов.

Вывод ресурса BlockDevice из LVMVolumeGroup

1. Чтобы исключить конкретный BlockDevice из состава Volume Group, отредактируйте селектор `spec.blockDeviceSelector` ресурса LVMVolumeGroup или удалите соответствующий лейбл у ресурса BlockDevice.
2. Вручную перенесите данные на узле (если в Volume Group есть тома) следующей командой:

```
pvmove <исходное-устройство>
```

3. Удалите физический том из группы:

```
vgreduce <имя-VG> <исключаемое-устройство>
```

4. Окончательно уберите метаданные физического тома:

```
pvremove <исключаемое-устройство>
```



Если в процессе удаления остаются логические тома, удалите их заранее с помощью команды `lvremove`. Для защиты от непреднамеренного удаления можно использовать аннотацию `storage.deckhouse.io/deletion-protection` — до её снятия ресурс LVMVolumeGroup не будет удалён.

Защита от утечек данных между томами

При удалении файлов операционная система не стирает данные на диске: соответствующие блоки лишь помечаются как «свободные». Если при создании нового тома используются те же физические блоки, они могут содержать остаточные данные предыдущего пользователя.

Например:

- Пользователь #1 создал том из StorageClass #1 на узле #1 (вне зависимости от режима Block или Filesystem) и записал в него файлы.
- Пользователь #1 удалил файлы и удалил сам том.
- Блоки, которые занимал этот том, были помечены как свободные, но не перезаписаны.
- Пользователь #2 запросил новый том из StorageClass #1 на том же узле #1 в блоковом режиме.
- Существует риск, что некоторые или все ранее освобождённые блоки будут вновь выделены пользователю #2.
- В результате пользователь #2 может получить доступ к данным пользователя #1.

Thick volume

Для обеспечения конфиденциальности используется параметр `volumeCleanup` в конфигурации `PersistentVolume`. Доступные режимы:

- Параметр не задан — при удалении тома никакие дополнительные операции не выполняются, блоки остаются неинициализированными, и данные могут быть доступны следующему пользователю.
- `RandomFillSinglePass` — том перезаписывается однократно случайными данными (рекомендуется избегать на твердотельных накопителях из-за износа).
- `RandomFillThreePass` — том трижды перезаписывается случайными данными (максимальная надёжность, но высокая нагрузка на устройство).
- `Discard` — вызовом TRIM все блочные устройства помечаются как свободные. Эффективно на большинстве современных твердотельных накопителей, но зависит от поддержки контроллера SSD (DRAT/RZAT).

Использование режима Discard

Большинство современных твердотельных накопителей поддерживает команду `discard` (TRIM), которая помечает освобождённые блоки как «ноль» и при последующем чтении не возвращает старые данные. Это делает опцию `Discard` наиболее надёжным способом предотвратить утечки данных на твердотельных накопителях. Однако следует учитывать следующие нюансы:

- Команда `discard` инициирует очистку ячеек, но фактическое обнуление выполняется контроллером накопителя в фоновом режиме. До завершения этой операции некоторые блоки могут содержать старые данные.
- Не все твердотельные накопители очищают конкретный физический блок по одиночке: многие устройства работают на уровне страниц и более крупных блоков, поэтому отдельные логические блоки могут оставаться неочищенными до тех пор, пока вся страница не будет полностью обработана.
- Чтобы быть уверенным, что после `discard` блоки действительно читаются как нули, диск должен поддерживать DRAT (Deterministic TRIM) — предсказуемое поведение команды `discard` и RZAT (Deterministic Read Zero after TRIM) — гарантированное чтение «нулей» после `discard`. Если устройство не имеет подтверждённой поддержки DRAT и RZAT, нет гарантии, что освобождённые блоки больше никогда не вернут прежние данные.
- Даже при заявленной поддержке DRAT/RZAT производители могут не соблюдать спецификации полностью. При отсутствии независимых тестов или сертификации твердотельных накопителей рекомендуется воздержаться от использования режима `Discard` для критичных задач, связанных с безопасностью данных.



Если накопитель не подтверждён как поддерживающий DRAT и RZAT (то есть не прошёл проверку на детерминированную очистку и чтение обнулённых блоков), использование опции `Discard` **не рекомендуется**, поскольку остаётся риск восстановления старых данных.

Thin volume

При освобождении блока `thin volume` команда `discard` из гостевой ОС передаётся непосредственно на устройство хранения. Если используется механический жёсткий диск или твердотельный накопитель без поддержки `discard`, освобождаемые блоки в `thin pool` остаются с предыдущими данными до момента их физической перезаписи. Однако клиентам предоставляются только `thin volume`, а не доступ к самому `thin pool`. При следующем выделении блока `thin-LV` LVM гарантированно заполняет соответствующий блок `thin pool` нулями (`zero-fill`), благодаря параметру `thin_pool_zero=1`. Это обеспечивает отсутствие остаточных данных между разными `thin volume` и предотвращает утечки информации между клиентами.

Дополнительные ресурсы

- Кастомные ресурсы модуля `sds-node-configurator`

Сценарии конфигурации

i Работоспособность гарантируется только при использовании стоковых ядер, поставляемых вместе с поддерживаемыми дистрибутивами. При использовании нестандартных ядер или дистрибутивов поведение может быть непредсказуемым.

Клонирование виртуальных машин

При создании виртуальных машин методом клонирования замените UUID групп томов (VG), выполнив:

```
vgchange -u
```

Команда сгенерирует новые UUID для всех VG на виртуальной машине. При необходимости команду можно добавить в скрипт `cloud-init`.

 Сменить UUID можно только если в группе нет активных логических томов (LV). Деактивировать их можно так:

```
lvchange -an <VG_or_LV_NAME>
```

Здесь `<VG_or_LV_NAME>` — название VG для деактивации всех томов в группе или название LV для деактивации конкретного тома.

Способы и сценарии конфигурации дисковой подсистемы узлов

Дисковую подсистему каждого узла можно организовать двумя способами — в зависимости от того, одинаковые ли диски установлены в сервере:

- хранилище с одинаковыми дисками — все диски в узле одного типа и размера;
- комбинированное хранилище — в узле установлены диски разных типов (например, SSD + HDD).

Для каждого из способов конфигурации дисковой подсистемы на узлах существует два сценария конфигурации:

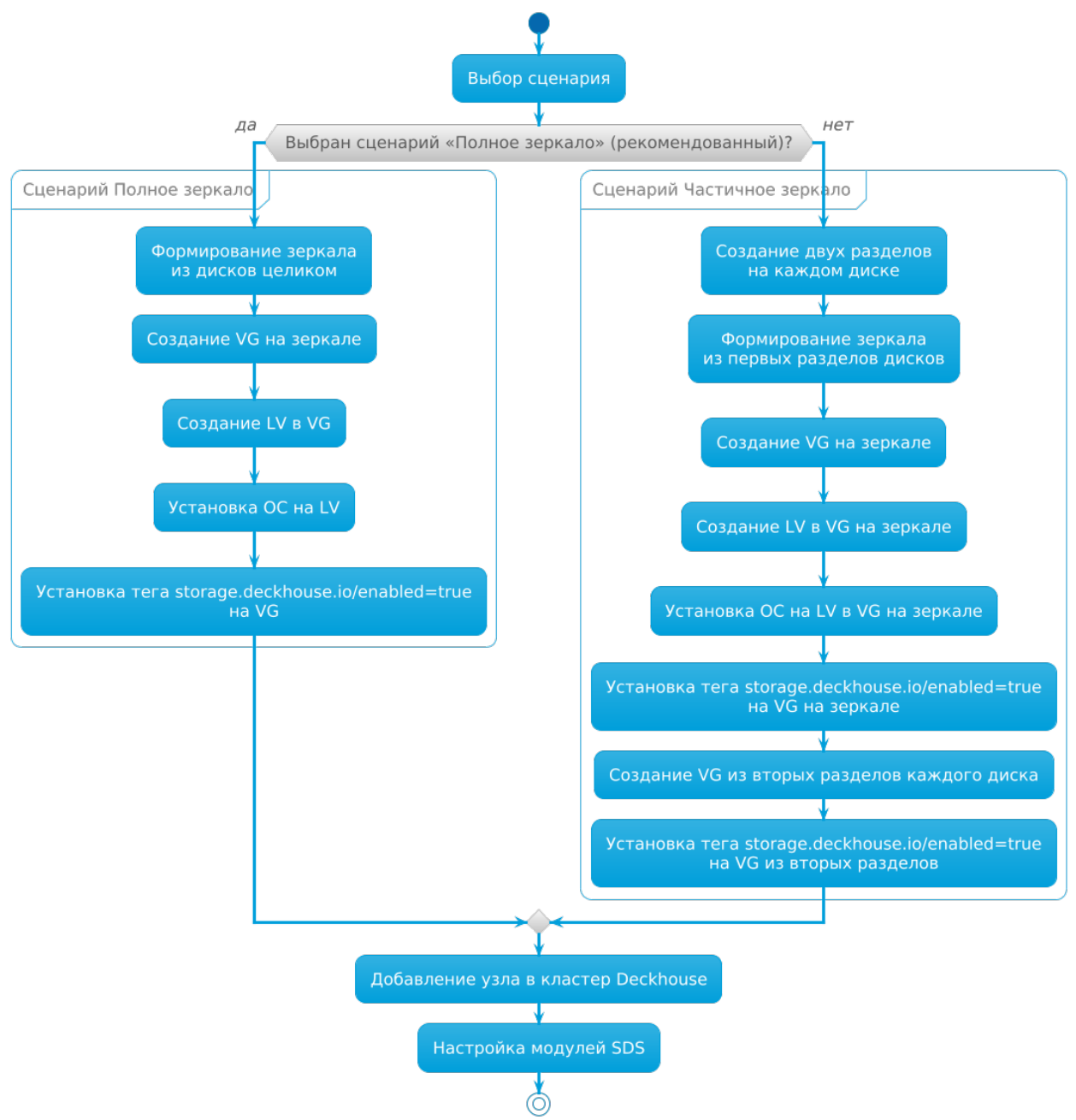
- **Полное зеркало** — рекомендуемый, надёжный и самый простой.
- **Частичное зеркало** — более гибкий, но требует аккуратности.

Особенности, плюсы и минусы сценариев приведены в таблице:

Сценарий конфигурации	Особенности реализации	Плюсы	Минусы
Полное зеркало	• Диски не делятся на разделы, создаётся зеркало из дисков целиком • Единая VG для ОС и данных	• Высокая надёжность • Простота настройки и эксплуатации • Гибкое распределение ресурсов между SDS	• Избыточное потребление диска при репликации SDS

Сценарий конфигурации	Особенности реализации	Плюсы	Минусы
Частичное зеркало	• Диски делятся на 2 раздела • Из первых разделов каждого диска создаётся зеркало, на нём создаётся VG под ОС • Из вторых разделов создаётся VG для данных без зеркалирования	• Надёжное хранение • Максимальная эффективность использования пространства	• Сложная настройка • Трудности с перераспределением места между safe- и unsafe-разделами

Различия в порядке конфигурации дисковой подсистемы в зависимости от выбранного сценария конфигурации изображены на схеме:



Хранилище с одинаковыми дисками

Полное зеркало

Мы рекомендуем использовать этот сценарий конфигурации, поскольку он достаточно надёжен и прост в настройке.

Чтобы настроить узел по этому сценарию, выполните следующее:

1. Создайте зеркало из всех дисков (аппаратно или программно). Это зеркало будет использоваться одновременно для корневой системы и для данных.
2. Установите операционную систему:
 - создайте VG с именем `main` на зеркале;
 - создайте LV с именем `root` в VG `main`;
 - установите операционную систему на LV `root`.
3. Установите тег `storage.deckhouse.io/enabled=true` для VG `main`, используя следующую команду:

```
vgchange main --addtag storage.deckhouse.io/enabled=true
```

4. Добавьте подготовленный узел в кластер Deckhouse.

Если узел подходит под `nodeSelector`, который указан в `spec.nodeSelector` модулей `sds-replicated-volume` или `sds-local-volume`, то агент `sds-node-configurator` обнаружит VG `main` и создаст ресурс `LVMVolumeGroup`. Его можно использовать в модулях `sds-local-volume` и `sds-replicated-volume`.

Пример настройки модулей SDS (одинаковые диски, «Полное зеркало»)

В этом сценарии три узла кластера Deckhouse сконфигурированы в режиме «Полное зеркало». После запуска автоматического обнаружения в кластере появятся три CRD-ресурса типа `LVMVolumeGroup` с автоматически сгенерированными именами.

Чтобы вывести список ресурсов `LVMVolumeGroup`, выполните следующую команду:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

В результате будет выведен следующий список:

NAME				THINPOOLS	CONFIGURATION	APPLIED	PHASE
NODE	SIZE	ALLOCATED	SIZE	VG	AGE		
vg-08d3730c-9201-428d-966c-45795cba55a6				0/0		True	Ready
worker-2	25596Mi	0		main	61s		
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d				0/0		True	Ready
worker-0	25596Mi	0		main	4m17s		
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7				0/0		True	Ready
worker-1	25596Mi	0		main	108s		

Настройка модуля sds-local-volume (одинаковые диски, «Полное зеркало»)

Чтобы сконфигурировать `sds-local-volume` в режиме «Полное зеркало», создайте ресурс `LocalStorageClass` и укажите в нём все обнаруженные `LVMVolumeGroup`. Это гарантирует, что VG с лейблом `main` будет доступен на каждом узле в модуле:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
    reclaimPolicy: Delete
    volumeBindingMode: WaitForFirstConsumer
EOF
```

Настройка модуля sds-replicated-volume (одинаковые диски, «Полное зеркало»)

Чтобы настроить модуль `sds-replicated-volume` по сценарию «Полное зеркало», выполните следующее:

1. Создайте ресурс `ReplicatedStoragePool` и добавьте в него все ресурсы `LVMVolumeGroup`, чтобы VG `main` использовалась на всех узлах в модуле `sds-replicated-volume`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

2. Создайте ресурс `ReplicatedStorageClass` и в поле `storagePool` укажите имя созданного ранее ресурса `ReplicatedStoragePool`:

```

d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # Если указать данную топологию, в кластере не должно быть
зон (узлов с лейблами topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF

```

Частичное зеркало



- Не используйте разделы с одинаковыми PARTUUID.
- Изменение PARTUUID раздела, на котором уже создан VG, не поддерживается.
- Для таблицы разделов рекомендуется использовать GPT — в MBR PARTUUID псевдослучайный и содержит номер раздела, а также отсутствует поддержка PARTLABEL, который может пригодиться для идентификации разделов в Deckhouse.

В данном сценарии используются два раздела на каждом диске:

- раздел для корневой системы и хранения данных SDS, которые не реплицируются;
- раздел для данных SDS, которые реплицируются.

Первый раздел каждого диска используется для создания зеркала, а второй — для создания отдельной VG без зеркалирования. Это позволяет максимально эффективно использовать место на диске.

Чтобы настроить узел по сценарию «Частичное зеркало», выполните следующее:

1. При установке операционной системы:
 - создайте по два раздела на каждом диске;
 - соберите зеркало из первых разделов на каждом диске;
 - создайте VG с именем `main-safe` на зеркале;
 - создайте LV с именем `root` в VG `main-safe`;
 - установите операционную систему на LV `root`.
2. Установите тег `storage.deckhouse.io/enabled=true` для VG `main-safe`, используя следующую команду:

```
vgchange main-safe --addtag storage.deckhouse.io/enabled=true
```

3. Создайте VG с именем `main-unsafe` из вторых разделов каждого диска.
4. Установите тег `storage.deckhouse.io/enabled=true` для VG `main-unsafe`, используя следующую команду:

```
vgchange main-unsafe --addtag storage.deckhouse.io/enabled=true
```

5. Добавьте подготовленный узел в кластер Deckhouse.

Если узел подходит под `nodeSelector`, который указан в `spec.nodeSelector` модулей `sds-replicated-volume` или `sds-local-volume`, то на этом узле запустится агент модуля `sds-node-configurator`, который определит VG `main-safe` и `main-unsafe` и добавит соответствующие этим VG ресурсы `LVMVolumeGroup` в кластер Deckhouse. Далее ресурсы `LVMVolumeGroup` можно использовать для создания томов в модулях `sds-replicated-volume` или `sds-local-volume`.

Пример настройки модулей SDS (одинаковые диски, «Частичное зеркало»)

В данном примере предполагается, что вы настроили три узла по сценарию «Частичное зеркало». В кластере Deckhouse при этом появятся шесть ресурсов `LVMVolumeGroup` со случайно сгенерированными именами. В будущем добавится возможность указывать имя для ресурсов `LVMVolumeGroup`, которые создаются в процессе автоматического обнаружения VG, с помощью тега `LVM` с желаемым именем ресурса.

Чтобы вывести список ресурсов `LVMVolumeGroup`, выполните следующую команду:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

В результате будет выведен следующий список:

NAME	SIZE	ALLOCATED	SIZE	VG	THINPOOLS	CONFIGURATION	APPLIED	PHASE
vg-08d3730c-9201-428d-966c-45795cba55a6					0/0	True		Ready
worker-2	25596Mi	0		main-safe		61s		
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d					0/0	True		Ready
worker-0	25596Mi	0		main-safe		4m17s		
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7					0/0	True		Ready
worker-1	25596Mi	0		main-safe		108s		
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91					0/0	True		Ready
worker-2	25596Mi	0		main-unsafe		61s		
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c					0/0	True		Ready
worker-0	25596Mi	0		main-unsafe		4m17s		
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2					0/0	True		Ready
worker-1	25596Mi	0		main-unsafe		108s		

Настройка модуля sds-local-volume (одинаковые диски, «Частичное зеркало»)

Чтобы настроить модуль `sds-local-volume` по сценарию «Частичное зеркало», создайте ресурс `LocalStorageClass` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-local-volume` использовалась только VG `main-safe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Настройка модуля sds-replicated-volume (одинаковые диски, «Частичное зеркало»)

Чтобы настроить модуль `sds-replicated-volume` по сценарию «Частичное зеркало», выполните следующее:

1. Создайте ресурс `ReplicatedStoragePool` с именем `data-safe` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-replicated-volume` в `ReplicatedStorageClass` с параметром `replication: None` использовалась только VG `main-safe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-safe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

2. Создайте ресурс `ReplicatedStoragePool` с именем `data-unsafe` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-replicated-volume` в `ReplicatedStorageClass` с параметром `replication: Availability` или `replication: ConsistencyAndAvailability` использовалась только VG `main-unsafe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-unsafe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
    - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
    - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
EOF
```

3. Создайте ресурс `ReplicatedStorageClass` и в поле `storagePool` укажите имя созданных ранее ресурсов `ReplicatedStoragePool`, чтобы на всех узлах использовались VG `main-safe` и `main-unsafe`:

```

d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data-safe # Обратите внимание, что из-за replication: None для
этого ресурса используется data-safe; следовательно, репликация данных для
постоянных томов (PV), созданных с этим StorageClass, проводиться не будет.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # Если указать данную топологию, в кластере не должно быть
зон (узлов с лейблами topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data-unsafe # Обратите внимание, что из-за replication:
Availability для этого ресурса используется data-unsafe; следовательно, будет
проводиться репликация данных для PV, созданных с этим StorageClass.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data-unsafe # Обратите внимание, что из-за replication:
ConsistencyAndAvailability для этого ресурса используется data-unsafe;
следовательно, будет проводиться репликация данных для PV, созданных с этим
StorageClass.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF

```

Комбинированное хранилище

Комбинированное хранилище предполагает одновременное использование на узле дисков разных типов.

В ситуации, когда для создания хранилища комбинируются диски разных типов, мы рекомендуем сделать зеркало из дисков одного типа и установить на него операционную систему по сценарию «Полное зеркало», но не использовать для SDS.

Для SDS используйте диски других типов (далее — дополнительные диски), отличающихся от тех, что используются для зеркала под операционную систему.

Рекомендации по использованию дополнительных дисков в зависимости от их типа:

Тип диска	Рекомендуемые цели использования
NVMe SSD	Создание томов, требующих высокой производительности
SATA SSD	Создание томов, не требующих высокой производительности
HDD	Создание томов, не требующих высокой производительности

Дополнительные диски можно настраивать по любому из сценариев «Полное зеркало» либо «Частичное зеркало».

Ниже будет рассмотрен процесс настройки дополнительных дисков на примере следующих типов:

- NVMe SSD;
- SATA SSD;
- HDD.

Настройка дополнительных дисков (Полное зеркало)



Ниже описан порядок действий по настройке дополнительных дисков для случая первичного развертывания и конфигурирования кластера при подключении к узлам по SSH. Если у вас уже есть работающий кластер и вы добавляете на его узлы дополнительные диски, рекомендуется создавать и настраивать VG с помощью ресурса LVMVolumeGroup, вместо выполнения на узле приведенных ниже команд.

Чтобы настроить дополнительные диски на узле по сценарию «Полное зеркало», выполните следующее:

1. Соберите зеркало из всех дополнительных дисков определенного типа целиком (аппаратно или программно).
2. Создайте VG с именем `<vg-name>` на зеркале.
3. Установите тег `storage.deckhouse.io/enabled=true` для VG `<vg-name>`, используя следующую команду:

```
vgchange <vg-name> --addtag storage.deckhouse.io/enabled=true
```



В примере выше замените `<vg-name>` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры имен VG для дополнительных дисков разных типов:

- `ssd-nvme` — для дисков NVMe SSD;
- `ssd-sata` — для дисков SATA SSD;
- `hdd` — для дисков HDD.

Пример настройки модулей SDS (комбинированное хранилище, «Полное зеркало»)

В данном примере предполагается, что вы настроили три узла по сценарию «Полное зеркало». В кластере Deckhouse при этом появятся три ресурса LVMVolumeGroup со случайно сгенерированными именами. В будущем добавится возможность указывать имя для ресурсов LVMVolumeGroup, которые создаются в процессе автоматического обнаружения VG, с помощью тега `LVM` с желаемым именем ресурса.

Чтобы вывести список ресурсов LVMVolumeGroup, выполните следующую команду:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

В результате будет выведен список вида:

NAME	SIZE	ALLOCATED	THINPOOLS	CONFIGURATION	APPLIED	PHASE
NODE	SIZE	SIZE	VG	AGE		
vg-08d3730c-9201-428d-966c-45795cba55a6			0/0	True		Ready
worker-2	25596Mi	0	<vg-name>	61s		
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d			0/0	True		Ready
worker-0	25596Mi	0	<vg-name>	4m17s		
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7			0/0	True		Ready
worker-1	25596Mi	0	<vg-name>	108s		

где `<vg-name>` — имя, присвоенное VG на зеркале на предыдущем шаге.

Настройка модуля sds-local-volume (комбинированное хранилище, «Полное зеркало»)

Чтобы настроить модуль `sds-local-volume` по сценарию «Полное зеркало», создайте ресурс `LocalStorageClass` и добавьте в него все ресурсы `LVMVolumeGroup`, чтобы VG `<vg-name>` использовалась на всех узлах в модуле `sds-local-volume`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i

В примере выше `<local-storage-class-name>` замените на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса `LocalStorageClass` для дополнительных дисков разных типов:

- `local-sc-ssd-nvme` — для дисков NVMe SSD;
- `local-sc-ssd-sata` — для дисков SATA SSD;
- `local-sc-ssd-hdd` — для дисков HDD.

Настройка модуля sds-replicated-volume (комбинированное хранилище, «Полное зеркало»)

Чтобы настроить модуль `sds-replicated-volume` по сценарию «Полное зеркало», выполните следующее:

1. Создайте ресурс `ReplicatedStoragePool` и добавьте в него все ресурсы `LVMVolumeGroup`, чтобы VG `<vg-name>` использовалась на всех узлах в модуле `sds-replicated-volume`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: <replicated-storage-pool-name>
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

i

В примере выше замените `<replicated-storage-pool-name>` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса `ReplicatedStoragePool` для дополнительных дисков разных типов:

- `data-ssd-nvme` — для дисков NVMe SSD;
- `data-ssd-sata` — для дисков SATA SSD;
- `data-hdd` — для дисков HDD.

2. Создайте ресурс ReplicatedStorageClass и в поле `storagePool` укажите имя созданного ранее ресурса ReplicatedStoragePool:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: <replicated-storage-pool-name>
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # Если указать данную топологию, в кластере не должно быть
зон (узлов с лейблами topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: <replicated-storage-pool-name>
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: <replicated-storage-pool-name>
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Настройка дополнительных дисков (Частичное зеркало)



- Не используйте разделы с одинаковыми PARTUUID.
- Изменение PARTUUID раздела, на котором уже создан VG, не поддерживается.
- Для таблицы разделов рекомендуется использовать GPT — в MBR PARTUUID псевдослучайный и содержит номер раздела, а также отсутствует поддержка PARTLABEL, который может пригодиться для идентификации разделов в Deckhouse.



Ниже описан порядок действий по настройке дополнительных дисков для случая первичного развертывания и конфигурирования кластера при подключении к узлам по SSH. Если у вас уже есть работающий кластер и вы добавляете на его узлы дополнительные диски, рекомендуется создавать и настраивать VG с помощью ресурса LVMVolumeGroup, вместо выполнения на узле приведенных ниже команд.

В данном сценарии используются два раздела на каждом диске: один для хранения данных SDS, которые не реплицируются, и другой для данных SDS, которые реплицируются. Первый раздел каждого диска используется для создания зеркала, а второй — для создания отдельной VG без зеркалирования. Это позволяет максимально эффективно использовать место на диске.

Чтобы настроить узел с дополнительными дисками по сценарию «Частичное зеркало», выполните следующее:

1. Создайте по два раздела на каждом дополнительном диске.
2. Соберите зеркало из первых разделов на каждом диске.
3. Создайте VG с именем `<vg-name>-safe` на зеркале.

4. Создайте VG с именем `<vg-name>-unsafe` из вторых разделов каждого диска.
5. Установите тег `storage.deckhouse.io/enabled=true` для VG `<vg-name>-safe` и `<vg-name>-unsafe`, используя следующую команду:

```
vgchange <vg-name>-safe --addtag storage.deckhouse.io/enabled=true
vgchange <vg-name>-unsafe --addtag storage.deckhouse.io/enabled=true
```

i

В примере выше `<vg-name>` замените на информативный префикс, в зависимости от типа дополнительных дисков.

Примеры информативного префикса `<vg-name>` для дополнительных дисков разных типов:

- `ssd-nvme` — для дисков NVMe SSD;
- `ssd-sata` — для дисков SATA SSD;
- `hdd` — для дисков HDD.

Пример настройки модулей SDS (комбинированное хранилище, «Частичное зеркало»)

В данном примере предполагается, что вы настроили три узла по сценарию «Частичное зеркало». В кластере Deckhouse при этом появятся шесть ресурсов `LVMVolumeGroup` со случайно сгенерированными именами. В будущем добавится возможность указывать имя для ресурсов `LVMVolumeGroup`, которые создаются в процессе автоматического обнаружения VG, с помощью тега `LVM` с желаемым именем ресурса.

Чтобы вывести список ресурсов `LVMVolumeGroup`, выполните следующую команду:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

В результате будет выведен список вида:

NAME	SIZE	ALLOCATED	THINPOOLS	CONFIGURATION	APPLIED	PHASE
NODE	SIZE	SIZE	VG	AGE		
vg-08d3730c-9201-428d-966c-45795cba55a6	0/0		True			Ready
worker-2	25596Mi	0	<vg-name>-safe	61s		
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d	0/0		True			Ready
worker-0	25596Mi	0	<vg-name>-safe	4m17s		
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7	0/0		True			Ready
worker-1	25596Mi	0	<vg-name>-safe	108s		
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91	0/0		True			Ready
worker-2	25596Mi	0	<vg-name>-unsafe	61s		
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c	0/0		True			Ready
worker-0	25596Mi	0	<vg-name>-unsafe	4m17s		
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2	0/0		True			Ready
worker-1	25596Mi	0	<vg-name>-unsafe	108s		

где `<vg-name>` — префикс имени, присвоенного VG, созданным на предыдущем шаге.

Настройка модуля `sds-local-volume` (комбинированное хранилище, «Частичное зеркало»)

Чтобы настроить модуль `sds-local-volume` по сценарию «Частичное зеркало», создайте ресурс `LocalStorageClass` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-local-volume` использовалась только VG `<vg-name>-safe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i

В примере выше замените `<local-storage-class-name>` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса `LocalStorageClass` для дополнительных дисков разных типов:

- `local-sc-ssd-nvme` — для дисков NVMe SSD;
- `local-sc-ssd-sata` — для дисков SATA SSD;
- `local-sc-hdd` — для дисков HDD.

Настройка модуля `sds-replicated-volume` (комбинированное хранилище, «Частичное зеркало»)

Чтобы настроить модуль `sds-replicated-volume` по сценарию «Частичное зеркало», выполните следующее:

1. Создайте ресурс `ReplicatedStoragePool` с именем `data-<vg-name>-safe` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-replicated-volume` в `ReplicatedStorageClass` с параметром `replication: None` использовалась только VG `<vg-name>-safe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-<vg-name>-safe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

i

В примере выше замените `data-<vg-name>-safe` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса `ReplicatedStoragePool` для дополнительных дисков разных типов:

- `data-ssd-nvme-safe` — для дисков NVMe SSD;
- `data-ssd-sata-safe` — для дисков SATA SSD;
- `data-hdd-safe` — для дисков HDD.

2. Создайте ресурс `ReplicatedStoragePool` с именем `data-<vg-name>-unsafe` и добавьте в него ресурсы `LVMVolumeGroup`, чтобы на всех узлах в модуле `sds-replicated-volume` в `ReplicatedStorageClass` с параметром `replication: Availability` или `replication:`

ConsistencyAndAvailability использовалась только VG `<vg-name>-unsafe`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-<vg-name>-unsafe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
    - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
    - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
EOF
```

i

В примере выше замените `data-<vg-name>-unsafe` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса ReplicatedStoragePool для дополнительных дисков разных типов:

- `data-ssd-nvme-unsafe` — для дисков NVMe SSD;
- `data-ssd-sata-unsafe` — для дисков SATA SSD;
- `data-hdd-unsafe` — для дисков HDD.

3. Создайте ресурс ReplicatedStorageClass и в поле `storagePool` укажите имя созданных ранее ресурсов ReplicatedStoragePool, чтобы на всех узлах использовались VG `<vg-name>-safe` и `<vg-name>-unsafe`:

```

d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: data-<vg-name>-safe # Обратите внимание, что из-за replication:
None для этого ресурса используется data-<vg-name>-safe; следовательно,
репликация данных для PV, созданных с этим StorageClass, проводиться не будет.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # Если указать данную топологию, в кластере не должно быть
зон (узлов с лейблами topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: data-<vg-name>-unsafe # Обратите внимание, что из-за
replication: Availability для этого ресурса используется data-<vg-name>-unsafe;
следовательно, будет проводиться репликация данных для PV, созданных с этим
StorageClass.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: data-<vg-name>-unsafe # Обратите внимание, что из-за
replication: ConsistencyAndAvailability для этого ресурса используется data-
<vg-name>-unsafe; следовательно, будет проводиться репликация данных для PV,
созданных с этим StorageClass.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF

```

i

В примере выше замените `data-<vg-name>-unsafe` на информативное имя, в зависимости от типа дополнительных дисков.

Примеры информативных имен ресурса `ReplicatedStoragePool` для дополнительных дисков разных типов:

- `data-ssd-nvme-unsafe` — для дисков NVMe SSD;
- `data-ssd-sata-unsafe` — для дисков SATA SSD;
- `data-hdd-unsafe` — для дисков HDD.

Аналогичным образом замените `data-<vg-name>-safe`.

Примеры информативных имен ресурса `ReplicatedStoragePool` для дополнительных дисков разных типов:

- `data-ssd-nvme-safe` — для дисков NVMe SSD;
- `data-ssd-sata-safe` — для дисков SATA SSD;
- `data-hdd-safe` — для дисков HDD.

Дополнительные ресурсы

- Документация модуля `sds-local-volume`

- Документация модуля `sds-node-configurator`
- Документация модуля `sds-replicated-volume`

FAQ

i Работоспособность гарантируется только при использовании стоковых ядер, поставляемых вместе с поддерживаемыми дистрибутивами. При использовании нестандартных ядер или дистрибутивов поведение может быть непредсказуемым.

Причины, по которым в кластере не создаются ресурсы BlockDevice

Чаще всего ресурсы BlockDevice не создаются, потому что имеющиеся устройства не проходят фильтры контроллера. Убедитесь, что устройства удовлетворяют требованиям.

Причины, по которым в кластере не создаются ресурсы LVMVolumeGroup

- Отсутствие BlockDevice — контроллер не создаст LVMVolumeGroup, если в кластере нет ресурсов BlockDevice, указанных в её спецификации.
- Отсутствие тега — если ресурсы BlockDevice присутствуют, но LVMVolumeGroup отсутствует, проверьте, что у соответствующей LVM-группы на узле задан тег `storage.deckhouse.io/enabled=true`.

Причины, по которым после удаления LVMVolumeGroup ресурс и Volume Group остаются

Ситуация возможна в двух случаях:

1. В Volume Group имеются логические тома — контроллер не отвечает за удаление логических томов (Logical Volumes) на узле, поэтому, если в созданной посредством ресурса Volume Group есть какие-либо логические тома, необходимо удалить их вручную. После этого и ресурс LVMVolumeGroup, и сама Volume Group (включая физические тома) будут удалены автоматически.
2. Для ресурса установлена аннотация `storage.deckhouse.io/deletion-protection` — данная аннотация защищает ресурс от удаления, а вместе с ним и связанную Volume Group. Уберите аннотацию командой:

```
d8 k annotate lvg <имя-ресурса> storage.deckhouse.io/deletion-protection-
```

После этого ресурс LVMVolumeGroup и соответствующая Volume Group будут удалены автоматически.

Причины неуспешного создания Volume Group с помощью ресурса LVMVolumeGroup

Вероятнее всего ресурс не прошёл валидацию контроллера (в отличие от схемы Kubernetes). Причину можно узнать в поле `status.message` ресурса LVMVolumeGroup или в логах контроллера. Проверьте, что указанные BlockDevice соответствуют условиям:

- поле `consumable` установлено в `true`;
- для `spec.type: Local` все BlockDevice принадлежат одному узлу;
- используются актуальные имена ресурсов BlockDevice.

Поведение ресурса LVMVolumeGroup при отключении одного из устройств в Volume Group

Ресурс LVMVolumeGroup остаётся, пока существует одноимённая LVM-группа. При недоступности устройства группа переходит в ошибочное состояние — это отражается в поле `status`.

После восстановления устройства группа возвращается в статус `Healthy`, а статус ресурса обновляется автоматически.

Передача управления существующей на узле Volume Group

контроллеру

Добавьте тег `storage.deckhouse.io/enabled=true` нужной Volume Group:

```
vgchange myvg-0 --addtag storage.deckhouse.io/enabled=true
```

Контроллер создаст соответствующий ресурс `LVMVolumeGroup` и возьмёт группу под управление.

Отключение отслеживания LVM Volume Group контроллером

Удалите тег `storage.deckhouse.io/enabled=true`:

```
vgchange myvg-0 --deltag storage.deckhouse.io/enabled=true
```

Контроллер прекратит отслеживание и удалит связанный ресурс `LVMVolumeGroup`.

Причины автоматической установки тега storage.deckhouse.io/enabled=true для Volume Group

Контроллер добавляет тег при создании Volume Group через ресурс `LVMVolumeGroup`.

При миграции с модуля `linstor` на `sds-node-configurator` и `sds-replicated-volume` все теги `linstor-*` заменяются на `storage.deckhouse.io/enabled=true`, чтобы передать управление новой логике.

Использование ресурса LVMVolumeGroupSet для создания LVMVolumeGroup

Ресурс `LVMVolumeGroupSet` позволяет создавать `LVMVolumeGroup` на узлах на основе шаблонов. Сейчас поддерживается стратегия `PerNode` — по одному ресурсу на каждый узел, удовлетворяющий селектору.

Пример спецификации `LVMVolumeGroupSet`:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroupSet
metadata:
  name: my-lvm-volume-group-set
  labels:
    my-label: my-value
spec:
  strategy: PerNode
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
  lvmVolumeGroupTemplate:
    metadata:
      labels:
        my-label-for-lvg: my-value-for-lvg
    spec:
      type: Local
      blockDeviceSelector:
        matchLabels:
          status.blockdevice.storage.deckhouse.io/model: <model>
      actualVGNameOnTheNode: <имя-VG-на-узле>
```

Лейблы, добавляемые контроллером к ресурсам BlockDevice

- `status.blockdevice.storage.deckhouse.io/type` — тип LVM;
- `status.blockdevice.storage.deckhouse.io/fstype` — тип файловой системы;
- `status.blockdevice.storage.deckhouse.io/pvuuid` — UUID физического тома (PV);
- `status.blockdevice.storage.deckhouse.io/vguuid` — UUID группы томов (VG);
- `status.blockdevice.storage.deckhouse.io/partuuid` — UUID раздела;

- `status.blockdevice.storage.deckhouse.io/lvmvolumegroupname` — имя ресурса LVMVolumeGroup;
- `status.blockdevice.storage.deckhouse.io/actualvgnameonthenode` — имя LVM Volume Group на узле;
- `status.blockdevice.storage.deckhouse.io/wwn` — WWN (World Wide Name) устройства;
- `status.blockdevice.storage.deckhouse.io/serial` — серийный номер устройства;
- `status.blockdevice.storage.deckhouse.io/size` — размер устройства;
- `status.blockdevice.storage.deckhouse.io/model` — модель устройства;
- `status.blockdevice.storage.deckhouse.io/rota` — флаг ротационного устройства;
- `status.blockdevice.storage.deckhouse.io/hotplug` — возможность горячего подключения;
- `status.blockdevice.storage.deckhouse.io/machineid` — идентификатор машины, где установлено устройство.

Дополнительные ресурсы

- Кастомные ресурсы модуля `sds-node-configurator`

Настройка локального хранилища на основе LVM

Локальное хранилище снижает сетевые задержки и обеспечивает более высокую производительность по сравнению с удалёнными хранилищами, доступ к которым осуществляется по сети. Такой подход особенно эффективен в тестовых средах и EDGE-кластерах. Данная функциональность обеспечивается модулем `sds-local-volume`.

Настройка локального хранилища

Для корректной работы модуля `sds-local-volume` выполните следующие шаги:

1. Настройте LVMVolumeGroup. Перед созданием StorageClass необходимо создать ресурс LVMVolumeGroup модуля `sds-node-configurator` на узлах кластера.
2. Включите модуль `sds-node-configurator`. Убедитесь, что модуль включен **до** включения модуля `sds-local-volume`.
3. Создайте соответствующие StorageClass'ы. Создание StorageClass для CSI-драйвера `local.csi.storage.deckhouse.io` пользователем **запрещено**.

Модуль поддерживает два режима работы: LVM и LVMThin.

Быстрый старт

Все команды выполняются на машине с доступом к API Kubernetes и правами администратора.

Включение модулей

Включение модуля `sds-node-configurator`:

1. Создайте ресурс ModuleConfig для включения модуля:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

2. Дождитесь состояния модуля `Ready`. На этом этапе не требуется проверять поды в неймспейсе `d8-sds-node-configurator`.

```
d8 k get modules sds-node-configurator -w
```

Включение модуля `sds-local-volume`:

1. Активируйте модуль `sds-local-volume`. Пример ниже запускает модуль с настройками по умолчанию, что приведет к созданию служебных подов компонента `sds-local-volume` на всех узлах кластера:


```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  version: 1
EOF
```

2. Дождитесь состояния модуля `Ready`.

```
d8 k get modules sds-local-volume -w
```

3. Убедитесь, что в неймспейсах `d8-sds-local-volume` и `d8-sds-node-configurator` все поды находятся в статусе `Running` или `Completed` и запущены на всех узлах, где планируется использовать ресурсы LVM.

```
d8 k -n d8-sds-local-volume get pod -owide -w
d8 k -n d8-sds-node-configurator get pod -O wide -w
```

Подготовка узлов к созданию хранилищ

Для корректной работы хранилищ на узлах необходимо, чтобы поды `sds-local-volume-csi-node` были запущены на выбранных узлах.

По умолчанию эти поды запускаются на всех узлах кластера. Проверить их наличие можно с помощью команды:

```
d8 k -n d8-sds-local-volume get pod -owide
```

Размещение подов `sds-local-volume-csi-node` управляется специальными лейблами (`nodeSelector`). Эти лейблы задаются в параметре `spec.settings.dataNodes.nodeSelector` модуля.

Настройка хранилища на узлах

Для настройки хранилища на узлах необходимо создать группы томов LVM с использованием ресурсов `LVMVolumeGroup`. В данном примере создается хранилище `Thick`.



Перед созданием ресурса `LVMVolumeGroup` убедитесь, что на данном узле запущен под `sds-local-volume-csi-node`. Это можно сделать командой:

```
d8 k -n d8-sds-local-volume get pod -owide
```

1. Получите все ресурсы `BlockDevice`, которые доступны в вашем кластере:

```
d8 k get bd
```

Пример вывода:

NAME PATH	NODE	CONSUMABLE	SIZE
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa 976762584Ki /dev/nvme1n1	worker-0	false	
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd 894006140416 /dev/nvme0n1p6	worker-0	false	
dev-7e4df1dddf2a1b05a79f9481cdf56d29891a9f9d0 976762584Ki /dev/nvme1n1	worker-1	false	
dev-b103062f879a2349a9c5f054e0366594568de68d 894006140416 /dev/nvme0n1p6	worker-1	false	
dev-53d904f18b912187ac82de29af06a34d9ae23199 976762584Ki /dev/nvme1n1	worker-2	false	
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1 894006140416 /dev/nvme0n1p6	worker-2	false	

2. Создайте ресурс LVMVolumeGroup для узла `worker-0`:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-0" # Подходит любое допустимое имя ресурса в
Kubernetes. Это имя ресурса LVMVolumeGroup будет использоваться для создания
LocalStorageClass в будущем.
spec:
  type: Local
  local:
    nodeName: "worker-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
          - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
  actualVGNameOnTheNode: "vg-1" # Имя LVM VG, который будет создан из указанных
блочных устройств на узле.
EOF
```

3. Дождитесь, когда созданный ресурс LVMVolumeGroup перейдет в состояние `Ready`:

```
d8 k get lvg vg-1-on-worker-0 -W
```

Если ресурс перешел в состояние `Ready`, это значит, что на узле `worker-0` из блочных устройств `/dev/nvme1n1` и `/dev/nvme0n1p6` была создана LVM VG с именем `vg-1`.

4. Создайте ресурс LVMVolumeGroup для узла `worker-1`:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-1"
spec:
  type: Local
  local:
    nodeName: "worker-1"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0
          - dev-b103062f879a2349a9c5f054e0366594568de68d
    actualVGNameOnTheNode: "vg-1"
EOF
```

5. Дождитесь, когда созданный ресурс LVMVolumeGroup перейдет в состояние Ready:

```
d8 k get lvg vg-1-on-worker-1 -W
```

Если ресурс перешел в состояние Ready, это значит, что на узле worker-1 из блочного устройства /dev/nvme1n1 и /dev/nvme0n1p6 была создана LVM VG с именем vg-1.

6. Создайте ресурс LVMVolumeGroup для узла worker-2:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-2"
spec:
  type: Local
  local:
    nodeName: "worker-2"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-53d904f18b912187ac82de29af06a34d9ae23199
          - dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1
    actualVGNameOnTheNode: "vg-1"
EOF
```

7. Дождитесь, когда созданный ресурс LVMVolumeGroup перейдет в состояние Ready:

```
d8 k get lvg vg-1-on-worker-2 -W
```

Если ресурс перешел в состояние Ready, то это значит, что на узле worker-2 из блочного устройства /dev/nvme1n1 и /dev/nvme0n1p6 была создана LVM VG с именем vg-1.

8. Создайте ресурс LocalStorageClass:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Для thin volume:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
    thin:
      poolName: thin-1
    type: Thin
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i **Важно.** В LocalStorageClass с type: Thick нельзя использовать LocalVolumeGroup, содержащие хотя бы один thin pool.

9. Дождитесь, когда созданный ресурс LocalStorageClass перейдет в состояние `Created`:

```
d8 k get lsc local-storage-class -w
```

10. Проверьте, что был создан соответствующий StorageClass:

```
d8 k get sc local-storage-class
```

Если StorageClass с именем `local-storage-class` появился, значит настройка модуля `sds-local-volume` завершена. Теперь пользователи могут создавать PVC, указывая StorageClass с именем `local-storage-class`.

Выбор метода очистки тома после удаления PV

При удалении файлов операционная система не удаляет содержимое физически, а лишь помечает соответствующие блоки как «свободные». Если новый том получает физические блоки, ранее использовавшиеся другим томом, в них могут остаться данные предыдущего пользователя.

Это возможно, например, в таком случае:

- пользователь №1 разместил файлы в томе, запрошенном из StorageClass 1 и на узле 1 (неважно, в режиме «Block» или «Filesystem»);

- пользователь №1 удалил файлы и том;
- физические блоки, которые он занимал, становятся «свободными», но не затертыми;
- пользователь №2 запросил новый том из StorageClass 1 и на узле 1 в режиме «Block»;
- есть риск, что часть или все блоки, ранее занимаемые пользователем №1, будут снова выделены пользователю №2;
- в этом случае пользователь №2 имеет возможность восстановить данные пользователя №1.

Thick-тома

Для предотвращения утечек через thick-тома предусмотрен параметр `volumeCleanup`. Он позволяет выбрать метод очистки тома перед удалением PV.

Возможные значения:

- Параметр не задан — не выполнять никаких дополнительных действий при удалении тома. Данные могут оказаться доступными следующему пользователю;
- `RandomFillSinglePass` — том будет перезаписан случайными данными один раз перед удалением. Использовать эту опцию не рекомендуется для твердотельных накопителей, так как она уменьшает ресурс накопителя;
- `RandomFillThreePass` — том будет перезаписан случайными данными три раза перед удалением. Использовать эту опцию не рекомендуется для твердотельных накопителей, так как она уменьшает ресурс накопителя;
- `Discard` — все блоки тома будут отмечены как свободные с использованием системного вызова `discard` перед удалением. Эта опция имеет смысл только для твердотельных накопителей.

Большинство современных твердотельных накопителей гарантирует, что помеченный `discard` блок при чтении не вернет предыдущие данные. Это делает опцию `Discard` самым эффективным способом предотвращения утечек при использовании твердотельных накопителей.

Однако очистка ячейки — относительно долгая операция, поэтому выполняется устройством в фоне. К тому же, многие диски не могут очищать индивидуальные ячейки, а только группы (страницы). Из-за этого не все накопители гарантируют немедленную недоступность освобожденных данных. Кроме того, не все накопители, гарантирующие это, держат обещание. Если устройство не гарантирует Deterministic TRIM (DRAT), Deterministic Read Zero after TRIM (RZAT) и не является проверенным, то использовать его не рекомендуется.

Thin-тома

В момент освобождения блока thin-тома через `discard` гостевой операционной системы эта команда пересылается на устройство. В случае использования жесткого диска или при отсутствии поддержки `discard` со стороны твердотельного накопителя данные могут остаться на thin pool до нового использования такого блока. Однако, пользователям предоставляется доступ только к thin-томам, а не к самому thin pool. Они могут получить только том из пула, а для thin-томов производится обнуление блока thin pool при новом использовании, что предотвращает утечки между клиентами. Это гарантируется настройкой `thin_pool_zero=1` в LVM.

Системные требования и рекомендации

- Используйте стоковые ядра, поставляемые вместе с поддерживаемыми дистрибутивами.
- Не используйте другой SDS (Software defined storage) для предоставления дисков SDS Deckhouse.

Дополнительные ресурсы

- Документация модуля `sds-local-volume`
- Документация модуля `sds-node-configurator`

Управление локальным хранилищем на основе LVM

Выбор конкретных узлов для использования модуля

Чтобы ограничить использование модуля определёнными узлами кластера, необходимо задать лейблы в поле `nodeSelector` в настройках модуля.

Для отображения и редактирования настроек модуля выполните команду:

```
d8 k edit mc sds-local-volume
```

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  settings:
    dataNodes:
      nodeSelector:
        my-custom-label-key: my-custom-label-value
status:
  message: ""
  version: "1"
```

Чтобы просмотреть текущие лейблы в поле `nodeSelector`, используйте следующую команду:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Пример вывода:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Модуль выбирает в качестве целевых только те узлы, у которых установлены все указанные в `nodeSelector` лейблы. Изменяя это поле, можно управлять списком узлов, на которых будет работать модуль.



В `nodeSelector` можно указать несколько лейблов. Однако для корректной работы модуля все эти лейблы должны присутствовать на каждом узле, на котором предполагается запуск `sds-local-volume-csi-node`.

После настройки лейблов убедитесь, что на целевых узлах запущены поды `sds-local-volume-csi-node`. Проверить их наличие можно командой:

```
d8 k -n d8-sds-local-volume get pod -owide
```

Проверка создания PVC на выбранном узле

Убедитесь, что на выбранном узле работает pod `sds-local-volume-csi-node`. Для этого выполните команду:

```
d8 k -n d8-sds-local-volume get po -owide
```

При отсутствии пода проверьте, что на узле установлены все лейблы, указанные в настройках модуля в поле `nodeSelector`. Подробнее о способах решения проблемы с отсутствием подов на нужном узле можно прочитать в этом разделе.

Вывод узла из-под управления модуля

Чтобы вывести узел из-под управления модуля, необходимо удалить лейблы, заданные в поле `nodeSelector` в настройках модуля `sds-local-volume`.

Для проверки текущих лейблов выполните команду:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Пример вывода:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Снимите указанные лейблы с узлов командой:

```
d8 k label node %node-name% %label-from-selector%-
```



После ключа лейбла необходимо указать знак минуса для её удаления.

После этого под `sds-local-volume-csi-node` должен быть удален с узла. Проверьте его состояние командой:

```
d8 k -n d8-sds-local-volume get po -owide
```

Если под остаётся после удаления лейбла, убедитесь, что лейблы из конфигурации `d8-sds-local-volume-controller-config` действительно удалены. Это можно проверить с помощью следующей команды:

```
d8 k get node %node-name% --show-labels
```

Если лейблы отсутствуют, проверьте, что на узле нет ресурсов `LVMVolumeGroup`, используемых ресурсами `LocalStorageClass`. Подробнее об этой проверке можно прочитать в этом разделе.



Обратите внимание, что для ресурсов `LVMVolumeGroup` и `LocalStorageClass`, из-за которых не удастся вывести узел из-под управления модуля, будет назначен лейбл `storage.deckhouse.io/sds-local-volume-candidate-for-eviction`.

На самом узле будет присутствовать лейбл `storage.deckhouse.io/sds-local-volume-need-manual-eviction`.

Проверка зависимых ресурсов LVMVolumeGroup на узле

Для проверки зависимых ресурсов выполните следующие шаги:

1. Отобразите имеющиеся ресурсы `LocalStorageClass`:

```
d8 k get lsc
```

2. Проверьте у каждого из них список используемых ресурсов `LVMVolumeGroup`.

Вы можете сразу отобразить содержимое всех `LocalStorageClass` ресурсов, выполнив команду:

```
d8 k get lsc -oyaml
```

Примерный вид `LocalStorageClass`:

```

apiVersion: v1
items:
- apiVersion: storage.deckhouse.io/v1alpha1
  kind: LocalStorageClass
  metadata:
    finalizers:
    - localstorageclass.storage.deckhouse.io
    name: test-sc
  spec:
    lvm:
      lvmVolumeGroups:
      - name: test-vg
      type: Thick
      reclaimPolicy: Delete
      volumeBindingMode: WaitForFirstConsumer
  status:
    phase: Created
  kind: List

```

Обратите внимание на поле `spec.lvm.lvmVolumeGroups` — именно в нем указаны используемые ресурсы `LVMVolumeGroup`.

3. Отобразите список существующих ресурсов `LVMVolumeGroup`:

```
d8 k get lvg
```

Примерный вывод `LVMVolumeGroup`:

NAME	HEALTH	NODE	SIZE	ALLOCATED SIZE	VG	AGE
lvg-on-worker-0	Operational	node-worker-0	40956Mi	0	test-vg	15d
lvg-on-worker-1	Operational	node-worker-1	61436Mi	0	test-vg	15d
lvg-on-worker-2	Operational	node-worker-2	122876Mi	0	test-vg	15d
lvg-on-worker-3	Operational	node-worker-3	307196Mi	0	test-vg	15d
lvg-on-worker-4	Operational	node-worker-4	307196Mi	0	test-vg	15d
lvg-on-worker-5	Operational	node-worker-5	204796Mi	0	test-vg	15d

4. Проверьте, что на узле, который вы собираетесь вывести из-под управления модуля, не присутствует какой-либо ресурс `LVMVolumeGroup`, используемый в ресурсах `LocalStorageClass`. Во избежание непредвиденной потери контроля за уже созданными с помощью модуля томами вручную удалите зависимые ресурсы, совершив необходимые операции над томом.

Оставшийся под `sds-local-volume-csi-node` после удаления лейблов

Если после удаления лейблов с узла под `sds-local-volume-csi-node` продолжает работать, это, вероятнее всего, связано с наличием на узле ресурсов `LVMVolumeGroup`, которые используются в одном из ресурсов `LocalStorageClass`. Процесс проверки описан выше.

Отсутствие служебных подов на нужном узле

Проблема может быть связана с некорректно установленными лейблами. Узлы, используемые модулем, определяются лейблами, заданными в поле `nodeSelector` в настройках модуля. Для просмотра текущих лейблов выполните:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Пример вывода:

```

nodeSelector:
  my-custom-label-key: my-custom-label-value

```

Дополнительно можно проверить селекторы, используемые модулем, в конфигурации секрета `d8-sds-local-volume-controller-config` в пространстве имён `d8-sds-local-volume`:


```
d8 k -n d8-sds-local-volume get secret d8-sds-local-volume-controller-config -O
jsonpath='{.data.config}' | base64 --decode
```

Пример вывода:

```
nodeSelector:
  kubernetes.io/os: linux
  my-custom-label-key: my-custom-label-value
```

В выводе данной команды должны быть указаны все лейблы из настроек модуля `data.nodeSelector`, а также `kubernetes.io/os: linux`.

Проверьте лейблы на нужном узле:

```
d8 k get node %node-name% --show-labels
```

При необходимости добавьте недостающие лейблы на желаемый узел:

```
d8 k label node %node-name% my-custom-label-key=my-custom-label-value
```

Если лейблы присутствуют, проверьте наличие лейбла `storage.deckhouse.io/sds-local-volume-node=` на узле. Если лейбл отсутствует, убедитесь, что работает `sds-local-volume-controller`, и ознакомьтесь с его логами:

```
d8 k -n d8-sds-local-volume get po -l app=sds-local-volume-controller
d8 k -n d8-sds-local-volume logs -l app=sds-local-volume-controller
```

Перемещение данных между PVC

Скопируйте следующий скрипт в файл `migrate.sh` на любом master-узле:

```
#!/bin/bash

ns=$1
src=$2
dst=$3

if [[ -z $3 ]]; then
  echo "You must give as args: namespace source_pvc_name destination_pvc_name"
  exit 1
fi

echo "Creating job yaml"
cat > migrate-job.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: migrate-pv-$src
  namespace: $ns
spec:
  template:
    spec:
      containers:
      - name: migrate
        image: debian
        command: [ "/bin/bash", "-c" ]
        args:
        -
          apt-get update && apt-get install -y rsync &&
          ls -lah /src_vol /dst_vol &&
          df -h &&
```

```

        rsync -avPS --delete /src_vol/ /dst_vol/ &&
        ls -lah /dst_vol/ &&
        du -shxc /src_vol/ /dst_vol/
    volumeMounts:
    - mountPath: /src_vol
      name: src
      readOnly: true
    - mountPath: /dst_vol
      name: dst
    restartPolicy: Never
    volumes:
    - name: src
      persistentVolumeClaim:
        claimName: $src
    - name: dst
      persistentVolumeClaim:
        claimName: $dst
    backoffLimit: 1
EOF

d8 k create -f migrate-job.yaml
d8 k -n $ns get jobs -o wide
kubectl_completed_check=0

echo "Waiting for data migration to be completed"
while [[ $kubectl_completed_check -eq 0 ]]; do
    d8 k -n $ns get pods | grep migrate-pv-$src
    sleep 5
    kubectl_completed_check=`d8 k -n $ns get pods | grep migrate-pv-$src | grep "Completed" |
wc -l`
done
echo "Data migration completed"

```

Для запуска скрипта выполните команду:

```
migrate.sh NAMESPACE SOURCE_PVC_NAME DESTINATION_PVC_NAME
```

Создание снимков томов

Подробную информацию о снимках и используемых ресурсах можно найти в документации Kubernetes.

1. Включите модуль `snapshot-controller`:

```

d8 k apply -f -<<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: snapshot-controller
spec:
  enabled: true
  version: 1
EOF

```

2. Теперь вы можете создавать снимки томов. Для этого выполните следующую команду с необходимыми параметрами:

```
d8 k apply -f -<<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: my-snapshot
  namespace: <name of the namespace where the PVC is located>
spec:
  volumeSnapshotClassName: sds-local-volume-snapshot-class
  source:
    persistentVolumeClaimName: <name of the PVC to snapshot>
EOF
```

Обратите внимание, что `sds-local-volume-snapshot-class` создается автоматически, и его `deletionPolicy` установлена в `Delete`, что означает, что ресурс `VolumeSnapshotContent` будет удален при удалении связанного ресурса `VolumeSnapshot`.

3. Чтобы проверить статус созданного снимка, выполните команду:

```
d8 k get volumesnapshot
```

Данная команда выведет список всех снимков и их текущий статус.

Дополнительные ресурсы

- Параметры модуля `sds-local-volume`
- Кастомные ресурсы модуля `sds-local-volume`
- Кастомные ресурсы модуля `sds-node-configurator`
- Документация модуля `snapshot-controller`

Настройка реплицируемого хранилища на основе DRBD

Репликация данных между несколькими узлами обеспечивает отказоустойчивость и доступность данных даже при сбоях в оборудовании или программном обеспечении одного из узлов. Это гарантирует, что данные сохранятся на других узлах, а доступ к ним будет непрерывным. Такая модель необходима для критически важных данных и распределенных инфраструктур с высокими требованиями к доступности и минимизации потерь при сбоях.

Для создания реплицируемых блочных объектов `StorageClass` на базе `Distributed Replicated Block Device (DRBD)`, распределенное реплицируемое блочное устройство) используется модуль `sds-replicated-volume`, который использует `LINSTOR` в качестве бэкенда.

Включение модуля

Обнаружение компонентов LVM

Перед тем как приступить к созданию объектов `StorageClass` на базе `LVM (Logical Volume Manager)`, необходимо найти доступные на узлах блочные устройства и группы томов и получить актуальную информацию об их состоянии. Для этого включите модуль `sds-node-configurator`:

```
d8 k apply -f -<<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, когда модуль `sds-node-configurator` перейдет в состояние `Ready`. Проверить состояние можно, выполнив следующую команду:

```
d8 k get modules sds-node-configurator -w
```

В результате будет выведена информация о модуле `sds-node-configurator`:

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
sds-node-configurator		Embedded	Available	True	True

Подключение DRBD

Чтобы включить модуль `sds-replicated-volume` с настройками по умолчанию, выполните команду:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

Это установит модуль ядра DRBD на всех узлах кластера, зарегистрирует CSI-драйвер и запустит служебные поды компонентов `sds-replicated-volume`.

Дождитесь, когда модуль `sds-replicated-volume` перейдет в состояние `Ready`. Проверить состояние можно, выполнив следующую команду:

```
d8 k get modules sds-replicated-volume -w
```

В результате будет выведена информация о модуле `sds-replicated-volume`:

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
sds-replicated-volume		Embedded	Available	True	True

Чтобы проверить, что в неймспейсах `d8-sds-replicated-volume` и `d8-sds-node-configurator` все поды в состоянии `Running` или `Completed` и запущены на всех узлах, где планируется использовать ресурсы DRBD, можно использовать команды:

```
d8 k -n d8-sds-replicated-volume get pod -w
d8 k -n d8-sds-node-configurator get pod -w
```

i Не рекомендуется настраивать бэкенд `LINSTOR` вручную, поскольку это может привести к ошибкам.

Преднастройка узлов

Создание групп томов LVM

Перед тем как приступить к настройке создания объектов `StorageClass`, необходимо объединить доступные на узлах блочные устройства в группы томов LVM. В дальнейшем группы томов будут использоваться для размещения `PersistentVolume`. Чтобы получить доступные блочные устройства, можно использовать ресурс `BlockDevice`, который отражает их актуальное состояние:

```
d8 k get bd
```

В результате будет выведен список доступных блочных устройств:

NAME PATH	NODE	CONSUMABLE	SIZE
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa /dev/nvme1n1	worker-0	false	976762584Ki
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd /dev/nvme0n1p6	worker-0	false	894006140416
dev-7e4df1dddf2a1b05a79f9481cdf56d29891a9f9d0 /dev/nvme1n1	worker-1	false	976762584Ki
dev-b103062f879a2349a9c5f054e0366594568de68d /dev/nvme0n1p6	worker-1	false	894006140416
dev-53d904f18b912187ac82de29af06a34d9ae23199 /dev/nvme1n1	worker-2	false	976762584Ki
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1 /dev/nvme0n1p6	worker-2	false	894006140416

В примере вывода перечислены шесть блочных устройств, расположенных на трёх узлах.

Чтобы объединить блочные устройства на одном узле, необходимо создать группу томов LVM с помощью ресурса LVMVolumeGroup. Для создания ресурса LVMVolumeGroup на узле `worker-0` примените следующий ресурс, предварительно заменив имена узла и блочных устройств на необходимые:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-on-worker-0"
spec:
  type: Local
  local:
    # Замените на имя своего узла, для которого вы создаете группу томов.
    nodeName: "worker-0"
    blockDeviceSelector:
      matchExpressions:
        - key: kubernetes.io/metadata.name
          operator: In
          values:
            # Замените на имена своих блочных устройств узла, для которого вы создаете
            # группу томов.
            - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
            - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
    # Имя группы томов LVM, которая будет создана из указанных выше блочных устройств
    # на выбранном узле.
    actualVGNameOnTheNode: "vg"
    # Раскомментируйте, если важно иметь возможность создавать thin pool.
    # thinPools:
    #   - name: thin-pool-0
    #     size: 70%
EOF
```

Дождитесь, когда созданный ресурс LVMVolumeGroup перейдет в состояние `Ready`. Чтобы проверить состояние ресурса, выполните следующую команду:

```
d8 k get lvg vg-on-worker-0 -w
```

В результате будет выведена информация о состоянии ресурса:

NAME ALLOCATED SIZE	THINPOOLS VG AGE	CONFIGURATION APPLIED	PHASE	NODE	SIZE
vg-on-worker-0 30064Mi	1/1 vg 1h	True	Ready	worker-0	360484Mi

Если ресурс перешел в состояние `Ready`, это значит, что на узле `worker-0` из блочных устройств `/dev/nvme1n1` и `/dev/nvme0n1p6` была создана группа томов LVM с именем `vg`.

Далее необходимо повторить создание ресурсов `LVMVolumeGroup` для оставшихся узлов (`worker-1` и `worker-2`), изменив в примере выше имя ресурса `LVMVolumeGroup`, имя узла и имена блочных устройств, соответствующих узлу.

Убедитесь, что группы томов LVM созданы на всех узлах, где планируется их использовать, выполнив следующую команду:

```
d8 k get lvg -w
```

В результате будет выведен список созданных групп томов:

NAME	THINPOOLS	CONFIGURATION	APPLIED	PHASE	NODE	SIZE
ALLOCATED SIZE	VG AGE					
vg-on-worker-0	0/0	True		Ready	worker-0	360484Mi
30064Mi	vg 1h					
vg-on-worker-1	0/0	True		Ready	worker-1	360484Mi
30064Mi	vg 1h					
vg-on-worker-2	0/0	True		Ready	worker-2	360484Mi
30064Mi	vg 1h					

Создание реплицированных `thick pool`

Теперь, когда на узлах созданы нужные группы томов LVM, необходимо объединить их в единое логическое пространство. Это можно сделать, объединив их в реплицированные пулы хранения в бэкенде `LINSTOR` через интерфейс в виде ресурса `ReplicatedStoragePool`.

Пулы хранения могут быть двух типов: LVM (`thick`) и LVMThin (`thin`). `Thick pool` обладает высокой производительностью, сравнимой с производительностью накопителя, но не позволяет использовать снимки. Пример создания реплицированного `thick pool`:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thick-pool
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
    - name: vg-1-on-worker-1
    - name: vg-1-on-worker-2
EOF
```

Дождитесь, когда созданный ресурс `ReplicatedStoragePool` перейдет в состояние `Completed`. Чтобы проверить состояние ресурса, выполните следующую команду:

```
d8 k get rsp data -w
```

В результате будет выведена информация о состоянии созданного ресурса:

NAME	PHASE	TYPE	AGE
thick-pool	Completed	LVM	87d

Создание реплицированных `thin pool`

i Для работы с `thin pool` необходимо включить параметр `enableThinProvisioning`

В отличие от `thick pool`, `thin pool` позволяет использовать снимки, но обладает меньшей производительностью.

Созданные ранее ресурсы LVMVolumeGroup подходят для создания thick pool. Если вам важно иметь возможность создавать реплицированные thin pool, обновите конфигурацию ресурсов LVMVolumeGroup, добавив определение для thin pool:

```
d8 k patch lvgl vg-on-worker-0 --type='json' -p='[
  {
    "op": "add",
    "path": "/spec/thinPools",
    "value": [
      {
        "name": "thin-pool-0",
        "size": "70%"
      }
    ]
  }
]
```

В обновленной версии LVMVolumeGroup 70% доступного пространства будет использовано для создания thin pool. Оставшиеся 30% могут быть использованы для thick pool.

Повторите добавление thin pool для оставшихся узлов (`worker-1` и `worker-2`). Пример создания реплицированного thin pool:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thin-pool
spec:
  type: LVMThin
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-1
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-2
      thinPoolName: thin-pool-0
EOF
```

Дождитесь, когда созданный ресурс ReplicatedStoragePool перейдет в состояние `Completed`. Чтобы проверить состояние ресурса, выполните следующую команду:

```
d8 k get rsp data -w
```

В результате будет выведена информация о состоянии созданного ресурса:

NAME	PHASE	TYPE	AGE
thin-pool	Completed	LVMThin	87d

Создание объектов StorageClass

Создание объектов StorageClass осуществляется через ресурс ReplicatedStorageClass, который определяет конфигурацию для желаемого класса хранения. Ручное создание ресурса StorageClass без ReplicatedStorageClass может привести к нежелательному поведению.

Пример создания ресурса ReplicatedStorageClass на основе thick pool, PersistentVolumes которого будут размещены на группах томов на трех узлах:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-storage-class
spec:
  # Имя одного из пулов хранения, созданных ранее.
  storagePool: thick-pool
  # Режим поведения при удалении PVC.
  # Допустимые значения: "Delete", "Retain".
  # Подробнее в документации Kubernetes:
  https://kubernetes.io/docs/concepts/storage/persistent-volumes/#reclaiming
  reclaimPolicy: Delete
  # Реплики смогут размещаться на любых доступных узлах: не более одной реплики
  # определенного тома на один узел.
  # В кластере нет зон (нет узлов с лейблами topology.kubernetes.io/zone).
  topology: Ignored
  # Режим репликации, при котором том остается доступным для чтения и записи, даже
  # если одна из реплик тома становится недоступной.
  # Данные хранятся в трех экземплярах на разных узлах.
  replication: ConsistencyAndAvailability
EOF
```

Проверьте, что созданный ресурс ReplicatedStorageClass перешел в состояние **Created**, выполнив следующую команду:

```
d8 k get rsc replicated-storage-class -w
```

В результате будет выведена информация о созданном ReplicatedStorageClass:

NAME	PHASE	AGE
replicated-storage-class	Created	1h

Убедитесь, что был создан соответствующий StorageClass, выполнив следующую команду:

```
d8 k get sc replicated-storage-class
```

В результате будет выведена информация о созданном StorageClass:

NAME	PROVISIONER	RECLAIMPOLICY
VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
replicated-storage-class	local.csi.storage.deckhouse.io	Delete
WaitForFirstConsumer	true	1h

Если StorageClass с именем **replicated-storage-class** появился, значит настройка модуля **sds-replicated-volume** завершена. Теперь пользователи могут создавать PersistentVolume, указывая StorageClass с именем **replicated-storage-class**. При указанных выше настройках будет создаваться том с тремя репликами на разных узлах.

Дополнительные ресурсы

- Документация модуля **sds-node-configurator**
- Документация модуля **sds-replicated-volume**

Управление реплицируемым хранилищем на основе DRBD



Работоспособность гарантируется только при соблюдении системных требований. Использование в иных условиях возможно, однако стабильная работа не гарантируется.

Системные требования

i Применимо как к однозональным кластерам, так и к кластерам с использованием нескольких зон доступности.

- Используйте стоковые ядра, поставляемые вместе с поддерживаемыми дистрибутивами.
- Для сетевого соединения используйте инфраструктуру с пропускной способностью 10 Gbps или выше.
- Чтобы достичь максимальной производительности, сетевая задержка между узлами должна находиться в пределах 0,5–1 мс.
- Не используйте другой SDS (Software defined storage) для предоставления дисков SDS Deckhouse.

Рекомендации

- Не используйте RAID. Подробнее ниже.
- Используйте локальные физические диски. Подробнее ниже.
- Для стабильной работы кластера, но с ухудшением производительности, допустимая сетевая задержка между узлами не должна превышать 20 мс.

Получение информации об использовании пространства

Доступно два способа получения информации:

1. Через дашборд Grafana:

Перейдите в раздел «Dashboards» → «Storage» → «LINSTOR/DRBD». В правом верхнем углу отображается текущий уровень занятости пространства кластера.

i **Внимание.** Значение отражает состояние всего доступного пространства. При создании томов с двумя репликами полученные цифры следует делить на два, чтобы оценить, сколько томов реально можно разместить.

2. Через командный интерфейс:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list
```

i **Внимание.** Значение отражает состояние всего доступного пространства. При создании томов с двумя репликами полученные цифры следует делить на два, чтобы оценить, сколько томов реально можно разместить.

Назначение StorageClass по умолчанию

Для назначения StorageClass по умолчанию необходимо в кастомном ресурсе ReplicatedStorageClass в поле `spec.isDefault` указать значение `true`.

Добавление существующей LVMVolumeGroup

1. Присвойте Volume Group LVM-тег `storage.deckhouse.io/enabled=true`:

```
vgchange myvg-0 --add-tag storage.deckhouse.io/enabled=true
```

После этого Volume Group будет автоматически обнаружена, и для неё будет создан ресурс LVMVolumeGroup.

2. Полученный ресурс укажите в параметрах ReplicatedStoragePool в поле `spec.lvmVolumeGroups[].name`. Если используется пул LVMThin, дополнительно задайте имя в `spec.lvmVolumeGroups[].thinPoolName`.

Изменение ограничений DRBD-томов и портов кластера

Портовой диапазон по умолчанию для DRBD-ресурсов — TCP 7000–7999. Переопределить его можно с помощью настройки `drbdPortRange`, указав нужные значения `minPort` и `maxPort`.



После изменения параметров `drbdPortRange` перезапустите контроллер LINSTOR, чтобы новые настройки вступили в силу. При этом существующие DRBD-ресурсы сохраняют назначенные им порты.

Корректная перезагрузка узла с DRBD-ресурсами



Чтобы обеспечить стабильную работу модуля, не перезагружайте несколько узлов одновременно.

1. Выполните `drain` нужного узла:

```
d8 k drain test-node-1 --ignore-daemonsets --delete-emptydir-data
```

2. Убедитесь, что нет проблемных DRBD-ресурсов и ресурсов в состоянии `SyncTarget`. Чтобы проверить это, выполните следующую команду и проанализируйте вывод:

```
d8 k -n d8-sds-replicated-volume exec -t deploy/linstor-controller -- linstor r l --faulty
```

Если ресурсы в состоянии `SyncTarget` присутствуют, дождитесь завершения синхронизации или выполните корректирующие действия.

Пример вывода:

```
Defaulted container "linstor-controller" out of: linstor-controller, kube-rbac-proxy
+-----+
| ResourceName | Node | Port | Usage | Conns | State | CreatedOn |
|=====|
+-----+
```

3. Перезагрузите узел и дождитесь синхронизации всех DRBD-ресурсов, затем выполните `uncordon`:

```
d8 k uncordon test-node-1
node/test-node-1 uncordoned
```

При необходимости перезагрузки еще одного узла повторите алгоритм.

Перемещение ресурсов для освобождения места в Storage Pool

1. Просмотрите список Storage Pool на исходном узле, чтобы определить, в каком из них недостаточно свободного места:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list -n OLD_NODE
```

2. Определите, какие тома находятся в переполненном Storage Pool, чтобы понять, какие из них можно перенести:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor volume list -n OLD_NODE
```

3. Получите список ресурсов, к которым относятся эти тома, чтобы далее можно было инициировать их перенос:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource list-volumes
```

4. Создайте копии выбранных ресурсов на другом узле (не более 1-2 ресурсов одновременно):

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-and-i-understand-what-i-am-doing resource create NEW_NODE RESOURCE_NAME
```

5. Дождитесь завершения синхронизации ресурсов между узлами, чтобы убедиться в корректности переноса:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource-definition wait-sync RESOURCE_NAME
```

6. Удалите ресурсы с исходного узла, чтобы освободить место в переполненном Storage Pool:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-and-i-understand-what-i-am-doing resource delete OLD_NODE RESOURCE_NAME
```

Автоматизированное управление репликами и мониторинг состояния LINSTOR

Управление репликами и мониторинг состояния автоматизированы в скрипте `replicas_manager.sh`. Он проверяет доступность контроллера LINSTOR, выявляет неисправные или поврежденные ресурсы, создает резервные копии базы данных и управляет дисковыми репликами, включая настройку `TieBreaker` для кворума.

Чтобы проверить существование скрипта `replicas_manager.sh`, выполните команду на любом из master-узлов:

```
ls -l /opt/deckhouse/sbin/replicas_manager.sh
```

После запуска скрипт выполняет следующие действия:

- проверяет доступность контроллера и соединение с сателлитами;
- выявляет неисправные или поврежденные ресурсы;
- создает резервную копию базы данных;
- управляет количеством дисковых реплик, добавляя новые при необходимости;
- настраивает `TieBreaker` для ресурсов с двумя репликами;
- логирует все действия в файл `linstor_replicas_manager_<дата_и_время>.log`;
- предоставляет рекомендации по устранению проблем, таких как застрявшие реплики.

Переменные для настройки `replicas_manager.sh`:

- `NON_INTERACTIVE` — включает неинтерактивный режим;
- `TIMEOUT_SEC` — таймаут между попытками, в секундах (по умолчанию: 10);
- `EXCLUDED_RESOURCES_FROM_CHECK` — регулярное выражение для исключения ресурсов из проверок;
- `CHUNK_SIZE` — размер чанка для обработки ресурсов (по умолчанию: 10);
- `NODE_FOR_EVICT` — имя узла, исключаемого из создания реплик;
- `LINSTOR_NAMESPACE` — пространство имен Kubernetes (по умолчанию: `d8-sds-replicated-volume`);
- `DISKLESS_STORAGE_POOL` — пул для diskless-реплик (по умолчанию: `DfltDisklessStorPool`).

Вытеснение DRBD-ресурсов с узла

Вытеснение DRBD-ресурсов с узла выполняется скриптом `evict.sh`. Он может работать в одном из двух режимов:

- Удаление узла — создаются дополнительные реплики для каждого ресурса, после чего узел удаляется из LINSTOR и Kubernetes.
- Удаление ресурсов — создаются реплики для ресурсов, затем сами ресурсы удаляются из LINSTOR (узел остаётся в кластере).

Подготовка и запуск скрипта

Перед выполнением вытеснения:

1. Убедитесь в наличии скрипта на master-узле:

```
ls -l /opt/deckhouse/sbin/evict.sh
```

2. Исправьте ошибочные ресурсы:

```
d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor resource list --
faulty
```

3. Проверьте, что все поды в пространстве имён `d8-sds-replicated-volume` находятся в состоянии `Running`:

```
d8 k -n d8-sds-replicated-volume get pods | grep -v Running
```

Пример удаления узла из LINSTOR и Kubernetes

Запустите скрипт `evict.sh` на любом из master-узлов в интерактивном режиме, указав режим удаления `--delete-node`:

```
/opt/deckhouse/sbin/evict.sh --delete-node
```

Для неинтерактивного режима добавьте флаг `--non-interactive` и укажите имя узла. В этом режиме скрипт выполнит все действия без запроса подтверждения от пользователя:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-node --node-name "worker-1"
```

Пример удаления ресурсов с узла

Запустите скрипт `evict.sh` на любом из master-узлов в интерактивном режиме, указав режим удаления `--delete-resources-only`:

```
/opt/deckhouse/sbin/evict.sh --delete-resources-only
```

Для неинтерактивного режима добавьте флаг `--non-interactive` и укажите имя узла. В этом режиме скрипт выполнит все действия без запроса подтверждения от пользователя:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-resources-only --node-name
"worker-1"
```



По окончании работы скрипта узел остаётся в статусе `SchedulingDisabled`, а в LINSTOR выставляется свойство `AutoplaceTarget=false`. Это блокирует автоматическое создание новых ресурсов на узле.

Чтобы вновь разрешить размещение ресурсов, выполните:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor node set-property "worker-1" AutoplaceTarget
d8 k uncordon "worker-1"
```

Проверьте параметр `AutoplaceTarget` у всех узлов (поле `AutoplaceTarget` будет пустым у тех узлов, на которых разрешено размещать ресурсы LINSTOR):

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor node list -S AutoplaceTarget
```

Параметры скрипта `evict.sh`

- `--delete-node` — удаление узла из LINSTOR и Kubernetes с предварительным созданием дополнительных реплик для всех ресурсов, размещенных на узле.
- `--delete-resources-only` — удаление ресурсов с узла без удаления узла из LINSTOR и Kubernetes с предварительным созданием дополнительных реплик для всех ресурсов, размещенных на узле.

- `--non-interactive` — запуск скрипта в неинтерактивном режиме.
- `--node-name` — имя узла, с которого необходимо вытеснить ресурсы. Параметр обязателен для использования в режиме `--non-interactive`.
- `--skip-db-backup` — пропустить создание резервной копии БД LINSTOR перед выполнением операций.
- `--ignore-advise` — выполнить операции, несмотря на предупреждения команды `linstor advise resource`. Следует использовать, если скрипт был прерван и количество реплик у части ресурсов не соответствует указанному в `ReplicatedStorageClass`.
- `--exclude-resources-from-check` — исключить из проверки ресурсы, перечисленные через символ `|`.

Диагностика проблем

Проблемы могут возникнуть на разных уровнях работы компонентов. Ниже приведена шпаргалка для диагностики неисправностей томов в LINSTOR.

Ошибка запуска `linstor-node` при загрузке DRBD-модуля

1. Проверьте состояние подов `linstor-node`:

```
d8 k get pod -n d8-sds-replicated-volume -l app=linstor-node
```

2. Если некоторые поды находятся в состоянии `Init`, проверьте версию DRBD и логи `bashible` на узле:

```
cat /proc/drbd
journalctl -fu bashible
```

Наиболее вероятные причины:

- Загружена версия DRBDv8 вместо требуемой DRBDv9. Проверьте версию:

```
cat /proc/drbd
```

Если файл `/proc/drbd` отсутствует, значит модуль не загружен и проблема не в этом.

- Включён Secure Boot. Поскольку DRBD компилируется динамически (аналог `dkms`) без цифровой подписи, модуль не поддерживается при включённом Secure Boot.

Ошибка `FailedMount` при запуске пода

При зависании пода на стадии `ContainerCreating`

Если под завис на стадии `ContainerCreating`, а в выводе команды `d8 k describe pod` присутствуют ошибки аналогичные той, что представлена ниже, значит устройство смонтировано на одном из других узлов:

```
rpc error: code = Internal desc = NodePublishVolume failed for pvc-b3e51b8a-9733-4d9a-bf34-84e0fee3168d: checking
for exclusive open failed: wrong medium type, check device health
```

Проверьте, где используется устройство, с помощью следующей команды:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor resource list -r pvc-b3e51b8a-9733-4d9a-bf34-84e0fee3168d
```

Флаг `InUse` покажет, на каком узле используется устройство. На этом узле потребуется вручную отмонтировать диск.

Ошибки `Input/output error`

Такие ошибки обычно возникают на этапе создания файловой системы (`mkfs`). Проверьте `dmesg` на узле, где запускается под:

```
dmesg | grep 'Remote failed to finish a request within'
```

Если вывод содержит сообщения вида `Remote failed to finish a request within ...`, возможно, скорость дисковой подсистемы недостаточна для корректной работы DRBD.

После удаления ресурса `ReplicatedStoragePool` остаётся соответствующий ему `Storage Pool`

Это ожидаемое поведение. Модуль `sds-replicated-volume` не обрабатывает операции удаления ресурса `ReplicatedStoragePool`.

Ограничения на изменение spec `ReplicatedStorageClass`

Изменению подлежит только поле `isDefault`. Остальные параметры ресурса являются неизменяемыми — такое поведение является ожидаемым.

Удаление дочернего `StorageClass` при удалении `ReplicatedStorageClass`

Если `StorageClass` находится в статусе `Created`, его можно удалить. При наличии иных статусов потребуется восстановление ресурса или ручное удаление `StorageClass`.

Ошибки при создании `Storage Pool` или `StorageClass`

При временных внешних проблемах (например, при недоступности `kube-apiserver`) модуль автоматически повторит выполнение неудачной операции.

Ошибка “You’re not allowed to change state of linstor cluster manually”

Операции, которые требуют ручного вмешательства, в модуле `sds-replicated-volume` частично или полностью автоматизированы. Поэтому модуль `sds-replicated-volume` ограничивает список разрешённых команд в LINSTOR. Например, автоматизировано создание Tie-Breaker, — сам LINSTOR иногда их не создаёт для ресурсов с двумя репликами. Чтобы посмотреть список разрешённых команд, выполните:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -  
- linstor'  
linstor --help
```

Восстановление БД из резервной копии

Резервные копии ресурсов бэкенда хранятся в секретах в виде YAML-файлов, сегментированных для удобства восстановления. Резервное копирование происходит автоматически по расписанию.

Пример корректно сформированной резервной копии выглядит следующим образом:

linstor-20240425074718-backup-0	Opaque	1
28s	sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718	
linstor-20240425074718-backup-1	Opaque	1
28s	sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718	
linstor-20240425074718-backup-2	Opaque	1
28s	sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718	
linstor-20240425074718-backup-completed	Opaque	0
28s	<none>	

Резервная копия хранится закодированными сегментами в секретах вида `linstor-%date_time%-backup-{0..2}`. Секрет вида `linstor-%date_time%-backup-completed` не содержит данных, и служит маркером корректно отработавшего процесса резервного копирования.

Процесс восстановления резервной копии

1. Задайте переменные окружения:

```
NAMESPACE="d8-sds-replicated-volume"
BACKUP_NAME="linstor_db_backup"
```

2. Проверьте наличие резервных копий:

```
d8 k -n $NAMESPACE get secrets --show-labels
```

Пример вывода команды:

linstor-20240425072413-backup-0	Opaque	1	33m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413				
linstor-20240425072413-backup-1	Opaque	1	33m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413				
linstor-20240425072413-backup-2	Opaque	1	33m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413				
linstor-20240425072413-backup-completed	Opaque	0	33m	<none>
linstor-20240425072510-backup-0	Opaque	1	32m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510				
linstor-20240425072510-backup-1	Opaque	1	32m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510				
linstor-20240425072510-backup-2	Opaque	1	32m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510				
linstor-20240425072510-backup-completed	Opaque	0	32m	<none>
linstor-20240425072634-backup-0	Opaque	1	31m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634				
linstor-20240425072634-backup-1	Opaque	1	31m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634				
linstor-20240425072634-backup-2	Opaque	1	31m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634				
linstor-20240425072634-backup-completed	Opaque	0	31m	<none>
linstor-20240425072918-backup-0	Opaque	1	28m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918				
linstor-20240425072918-backup-1	Opaque	1	28m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918				
linstor-20240425072918-backup-2	Opaque	1	28m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918				
linstor-20240425072918-backup-completed	Opaque	0	28m	<none>
linstor-20240425074718-backup-0	Opaque	1	10m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718				
linstor-20240425074718-backup-1	Opaque	1	10m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718				
linstor-20240425074718-backup-2	Opaque	1	10m	sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718				
linstor-20240425074718-backup-completed	Opaque	0	10m	<none>

3. Каждая резервная копия имеет свой лейбл с временем создания. Выберите нужный и скопируйте лейбл в переменную окружения. В качестве примера используется лейбл самой актуальной копии из вывода выше:

```
LABEL_SELECTOR="sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718"
```

4. Создайте временный каталог для хранения частей архива:

```
TMPDIR=$(mktemp -d)
echo "Временный каталог: $TMPDIR"
```

5. Создайте пустой архив и объедините данные секретов в один файл:

```
COMBINED="${BACKUP_NAME}_combined.tar"
> "$COMBINED"
```

6. Получите список секретов по лейблу, дешифруйте данные и поместите данные резервной копии в архив:


```
MOBJECTS=$(d8 k get rsmb -l "$LABEL_SELECTOR" --sort-by=.metadata.name -o jsonpath="{.items[*].metadata.name}")

for MOBJECT in $MOBJECTS; do
  echo "Process: $MOBJECT"
  d8 k get rsmb "$MOBJECT" -o jsonpath="{.data}" | base64 --decode >> "$COMBINED"
done
```

7. Распакуйте архив, получив файлы резервной копии:

```
mkdir -p "./backup"
tar -xf "$COMBINED" -C ./backup --strip-components=2
```

8. Проверьте содержимое резервной копии:

```
ls ./backup
TMPDIR=$(mktemp -d)
echo "Временный каталог: $TMPDIR"
```

Пример вывода:

```
ebsremotes.yaml                                layerdrbdvolumedefinitions.yaml
layerwritecachevolumes.yaml propscontainers.yaml satellitescapacity.yaml
secidrolemap.yaml trackingdate.yaml
files.yaml layerdrbdvolumes.yaml
linstorremotes.yaml resourceconnections.yaml schedules.yaml
secobjectprotection.yaml volumeconnections.yaml
keyvaluestore.yaml layerluksvolumes.yaml
linstorversion.yaml resourcedefinitions.yaml secaccesstypes.yaml
secroles.yaml volumedefinitions.yaml
layerbcachevolumes.yaml layeropenflexresourcedefinitions.yaml
nodeconnections.yaml resourcegroups.yaml secacclmap.yaml
sectyperules.yaml volumegroups.yaml
layercachevolumes.yaml layeropenflexvolumes.yaml
nodenetinterfaces.yaml resources.yaml secconfiguration.yaml
sectypes.yaml volumes.yaml
layerdrbdresourcedefinitions.yaml layerresourceids.yaml
nodes.yaml rollback.yaml secdfltroles.yaml
spacehistory.yaml
layerdrbdresources.yaml layerstoragevolumes.yaml
nodestorpool.yaml s3remotes.yaml secidentities.yaml
storpooldefinitions.yaml
```

9. Восстановите необходимую сущность, применив соответствующий YAML-файл:

```
d8 k apply -f %something%.yaml
```

Либо примените bulk-apply, если нужно полное восстановление:

```
d8 k apply -f ./backup/
```

Отсутствие служебных подов sds-replicated-volume на выбранном узле

С высокой вероятностью проблемы связаны с лейблами на узлах.

- Проверьте `dataNodes.nodeSelector` в настройках модуля:

```
d8 k get mc sds-replicated-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```


- Проверьте селекторы, используемые `sds-replicated-volume-controller`:

```
d8 k -n d8-sds-replicated-volume get secret d8-sds-replicated-volume-controller-config -o
jsonpath='{.data.config}' | base64 --decode
```

- В секрете `d8-sds-replicated-volume-controller-config` должны быть селекторы, которые указаны в настройках модуля, а также дополнительно селектор `kubernetes.io/os: linux`.
- Проверьте, что на нужном узле есть все указанные в секрете `d8-sds-replicated-volume-controller-config` лейблы:

```
d8 k get node worker-0 --show-labels
```

- Если лейблов нет, их необходимо добавить через шаблоны в NodeGroup или на узел.
- Если лейблы есть, необходимо проверить, есть ли на нужном узле лейбл `storage.deckhouse.io/sds-replicated-volume-node=`. Если его нет, проверьте, запущен ли `sds-replicated-volume-controller` и если запущен, то проверьте его логи:

```
d8 k -n d8-sds-replicated-volume get po -l app=sds-replicated-volume-controller
d8 k -n d8-sds-replicated-volume logs -l app=sds-replicated-volume-controller
```

Дополнительная поддержка

Информация о причинах неудачных операций отображается в поле `status.reason` ресурсов `ReplicatedStoragePool` и `ReplicatedStorageClass`. При недостатке информации для диагностики рекомендуется обращаться к логам `sds-replicated-volume-controller`.

Миграция с модуля `linstor` на `sds-replicated-volume`

При миграции временно недоступны control plane LINSTOR и его CSI, что может повлиять на операции с PV (создание, расширение или удаление).



Миграция не затрагивает пользовательские данные, поскольку происходит перенос в новое пространство имён с добавлением компонентов управления томами.

Порядок действий для миграции

1. Убедитесь, что в бэкенде отсутствуют неисправные ресурсы. Команда должна выводить пустой список:

```
alias linstor='d8 k -n d8-linstor exec -ti deploy/linstor-controller --
linstor'
linstor resource list --faulty
```



Внимание. Перед миграцией все ресурсы должны быть исправны.

2. Выключите модуль `linstor`:

```
d8 k patch moduleconfig linstor --type=merge -p '{"spec": {"enabled": false}}'
```

3. Дождитесь, когда пространство имён `d8-linstor` будет удалено:

```
d8 k get namespace d8-linstor
```

4. Создайте ресурс ModuleConfig для `sds-node-configurator`:

```
d8 k apply -f -<<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

5. Дождитесь, когда модуль `sds-node-configurator` перейдет в состояние `Ready`:

```
d8 k get moduleconfig sds-node-configurator
```

6. Создайте ресурс ModuleConfig для `sds-replicated-volume`:

i **Внимание.** Если в настройках модуля `sds-replicated-volume` не будет указан параметр `settings.dataNodes.nodeSelector`, то значение для этого параметра при установке модуля `sds-replicated-volume` будет взято из модуля `linstor`. Если этот параметр не указан и там, то только в этом случае он останется пустым и все узлы кластера будут считаться узлами для хранения данных.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

7. Дождитесь, когда модуль `sds-replicated-volume` перейдет в состояние `Ready`:

```
d8 k get moduleconfig sds-replicated-volume
```

8. Проверьте настройки модуля `sds-replicated-volume`:

```
d8 k get moduleconfig sds-replicated-volume -oyaml
```

9. Дождитесь, пока все поды в пространстве имён `d8-sds-replicated-volume` и `d8-sds-node-configurator` перейдут в состояние `Ready` или `Completed`:

```
d8 k get po -n d8-sds-node-configurator
d8 k get po -n d8-sds-replicated-volume
```

10. Измените алиас к команде `linstor` и проверьте ресурсы:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-
controller -- linstor'
linstor resource list --faulty
```

Если неисправные ресурсы не обнаружены, значит миграция была успешной.

Миграция на ReplicatedStorageClass

StorageClass в данном модуле управляются через ресурс ReplicatedStorageClass. Вручную StorageClass создаваться не должны.

При миграции с модуля LINSTOR удалите старые StorageClass и создайте новые через ресурс ReplicatedStorageClass в соответствии с таблицей, представленной ниже.

Обратите внимание, что в старых StorageClass нужно смотреть опцию из секции `parameter` самого StorageClass, а указывать соответствующую опцию при создании нового необходимо в ReplicatedStorageClass.

Параметр StorageClass	ReplicatedStorageClass	Параметр по умолчанию	Примечания
linstor.csi.linbit.com/placementCount: "1"	replication: "None"		Будет создаваться одна реплика тома с данными
linstor.csi.linbit.com/placementCount: "2"	replication: "Availability"		Будут создаваться две реплики тома с данными
linstor.csi.linbit.com/placementCount: "3"	replication: "ConsistencyAndAvailability"	Да	Будут создаваться три реплики тома с данными
linstor.csi.linbit.com/storagePool: "name"	storagePool: "name"		Название используемого storage pool для хранения
linstor.csi.linbit.com/allowRemoteVolumeAccess: "false"	volumeAccess: "Local"		Запрещен удаленный доступ пода к томам с данными (только локальный доступ к диску в пределах узла)

Дополнительно можно задавать параметры:

- `reclaimPolicy` (Delete, Retain) — соответствует параметру `reclaimPolicy` у старого StorageClass;
- `zones` — перечисление зон, которые нужно использовать для размещения ресурсов (прямое указание названия зон в облаке). Обратите внимание, что удаленный доступ пода к тому с данными возможен только в пределах одной зоны;
- `volumeAccess` может принимать значения `Local` (доступ строго в пределах узла), `EventuallyLocal` (реплика данных будет синхронизироваться на узле с запущенным подом спустя некоторое время после запуска), `PreferablyLocal` (удаленный доступ пода к тому с данными разрешен, `volumeBindingMode: WaitForFirstConsumer`), `Any` (удаленный доступ пода к тому с данными разрешен, `volumeBindingMode: Immediate`);
- при необходимости использовать `volumeBindingMode: Immediate` нужно выставлять параметр ReplicatedStorageClass `volumeAccess` равным `Any`.

Миграция на ReplicatedStoragePool

Ресурс ReplicatedStoragePool предназначен для создания Storage Pool в бэкенде модуля. Рекомендуется создать этот ресурс даже для уже существующих Storage Pool и указать в этом ресурсе существующие LVMVolumeGroup. В этом случае контроллер увидит, что соответствующие Storage Pool созданы, и оставит их без изменений, а в поле `status.phase` созданного ресурса будет отображено значение `Created`.

Миграция с модуля sds-drbd на sds-replicated-volume

В процессе миграции будет недоступен управляющий слой модуля и его CSI. Это приведет к невозможности создания, расширения или удаления PV и создания или удаления подов, использующих PV DRBD на время проведения миграции.



Миграция не затронет пользовательские данные, поскольку выполняется в новое пространство имён, а управление томами будет передано новым компонентам, которые заменят функциональность прежнего модуля.

Порядок действий для миграции

1. Убедитесь, что в кластере отсутствуют неисправные DRBD-ресурсы. Команда должна выводить пустой список:

```
alias linstor='d8 k -n d8-sds-drbd exec -ti deploy/linstor-controller --
linstor'
linstor resource list --faulty
```

i **Внимание.** Перед миграцией все DRBD-ресурсы должны работать корректно.

2. Выключите модуль `sds-drbd`:

```
d8 k patch moduleconfig sds-drbd --type=merge -p '{"spec": {"enabled": false}}'
```

3. Дождитесь, когда пространство имён `d8-sds-drbd` будет удалено:

```
d8 k get namespace d8-sds-drbd
```

4. Создайте ресурс ModuleConfig для `sds-replicated-volume`:

i **Внимание.** Если в настройках модуля `sds-replicated-volume` не будет указан параметр `settings.dataNodes.nodeSelector`, то значение для этого параметра при установке модуля `sds-replicated-volume` будет взято из модуля `sds-drbd`. Если этот параметр не указан и там, то только в этом случае он останется пустым и все узлы кластера будут считаться узлами для хранения данных.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

5. Дождитесь, когда модуль `sds-replicated-volume` перейдет в состояние `Ready`:

```
d8 k get moduleconfig sds-replicated-volume
```

6. Проверьте настройки модуля `sds-replicated-volume`:

```
d8 k get moduleconfig sds-replicated-volume -oyaml
```

7. Дождитесь, пока все поды в пространстве имён `d8-sds-replicated-volume` перейдут в состояние `Ready` или `Completed`:

```
d8 k get po -N d8-sds-replicated-volume
```

8. Измените алиас к команде `linstor` и проверьте ресурсы DRBD:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-
controller -- linstor'
linstor resource list --faulty
```

Если неисправные ресурсы не обнаружены, значит миграция была успешной.

i **Внимание.** Ресурсы `DRBDStoragePool` и `DRBDStorageClass` в процессе будут автоматически мигрированы на `ReplicatedStoragePool` и `ReplicatedStorageClass`. Ручное вмешательство при этом не требуется.

Логика работы этих ресурсов не изменится. Однако, стоит проверить, не осталось ли в кластере ресурсов `DRBDStoragePool` или `DRBDStorageClass`. Если после миграции они существуют, обратитесь в техническую поддержку Deckhouse.

Причины отказа от RAID с `sds-replicated-volume`

Использование DRBD с более чем одной репликой уже обеспечивает функциональность сетевого RAID. Применение RAID локально может привести к следующим проблемам:

- В несколько раз увеличивает дополнительное потребление используемого пространства в случае использования RAID с избыточностью. Например, используется `ReplicatedStorageClass` с `replication`, выставленным в `ConsistencyAndAvailability`. При таких настройках DRBD будет сохранять данные в трех репликах (по одной реплике на три разных узла). Если на этих узлах будет использоваться RAID1, то для хранения 1 ГБ данных потребуется суммарно 6 ГБ места на дисках. RAID с избыточностью есть смысл использовать для упрощения обслуживания серверов в том случае, когда цена хранения не имеет значения. RAID1 в таком случае позволит менять диски на серверах без необходимости перемещения реплик данных с “проблемного” диска.
- В случае RAID0 прирост производительности будет незаметен, т. к. репликация данных будет осуществляться по сети и узким местом с высокой вероятностью будет именно сеть. Кроме того, уменьшение надежности хранилища на хосте потенциально будет приводить к недоступности данных, тк в DRBD переключение со сломавшейся реплики на здоровую происходит не мгновенно.

Рекомендации по использованию локальных дисков

DRBD использует сеть для репликации данных. При использовании NAS нагрузка на сеть будет расти кратно, так как узлы будут синхронизировать данные не только с NAS, но и между собой. Аналогично будет расти задержка на чтение или запись. NAS обычно предполагает использование RAID на своей стороне, что также увеличивает дополнительную нагрузку.

Ручной запуск процесса перевыпуска сертификатов

Несмотря на то, что процесс перевыпуска сертификатов автоматизирован, ручной перевыпуск всё ещё может понадобиться, так как его можно произвести в удобное временное окно, когда допустимо перезапустить объекты модуля. Автоматизированный перевыпуск не перезапускает объекты.

Чтобы вручную инициировать процесс продления сертификатов, создайте `ConfigMap` с именем `manualcertrenewal-trigger`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manualcertrenewal-trigger
  namespace: d8-sds-replicated-volume
```

Система остановит все необходимые объекты модуля, обновит сертификаты и затем снова их запустит.

Статус операции можно определить с помощью команды:

```
d8 k -n d8-sds-replicated-volume get cm manualcertrenewal-trigger -ojsonpath='{.data.step}'
```

Возможные статусы:

- `Prepared` — проверки состояния успешно пройдены, начато время простоя;
- `TurnedOffAndRenewedCerts` — система остановлена, сертификаты обновлены;
- `TurnedOn` — система снова запущена;
- `Done` — операция завершена и готова к повторению.

Сертификаты выдаются сроком на один год и помечаются как устаревающие за 30 дней до истечения срока действия. Система мониторинга оповещает о наличии устаревающих сертификатов с помощью алерта `D8LinstorCertificateExpiringIn30d`.

Чтобы повторить операцию, достаточно удалить лейбл с триггера с помощью команды:

```
d8 k -n d8-sds-replicated-volume label cm manualcertrenewal-trigger storage.deckhouse.io/sds-replicated-volume-manualcertrenewal-completed-
```

Дополнительные ресурсы

- Параметры модуля `sds-local-volume`
- Документация модуля `sds-node-configurator`
- Документация модуля `sds-replicated-volume`

Распределённое хранилище Ceph

Ceph — это масштабируемая распределённая система хранения с высокой доступностью и отказоустойчивостью. Deckhouse Kubernetes Platform (DKP) обеспечивает интеграцию Ceph-кластера при помощи модуля `csi-ceph`. Это даёт возможность динамически управлять хранилищем и использовать StorageClass на основе RADOS Block Device (RBD) или CephFS.

На этой странице представлены инструкции по подключению Ceph в Deckhouse, настройке аутентификации, созданию объектов StorageClass, а также проверке работоспособности хранилища.



Для работы со снимками требуется подключенный модуль `snapshot-controller`.

Миграция с модуля `ceph-csi`

При переключении на модуль `csi-ceph` с модуля `ceph-csi` выполняется автоматическая миграция, но её запуск требует предварительной подготовки:

1. Установите количество реплик в ноль для всех операторов (redis, clickhouse, kafka и др.).
Исключение: оператор `prometheus` будет отключён автоматически.
2. Отключите модуль `ceph-csi` и включите `csi-ceph`.
3. Дождитесь завершения операции. В логах DKP должно появиться сообщение “Finished migration from Ceph CSI module”.
4. Проверьте работоспособность. Для этого создайте тестовые поды и PVC для проверки CSI.
5. Верните операторы в рабочее состояние (установите количество реплик обратно).



Если Ceph StorageClass был создан не через ресурс `CephCSIDriver`, потребуется ручная миграция. Обратитесь в техподдержку.

Подключение к Ceph-кластеру

Для подключения Ceph-кластера следуйте пошаговым инструкциям ниже. Все команды выполняйте на машине с административным доступом к API Kubernetes.

1. Выполните команду для активации модуля `csi-ceph`:

```
d8 s module enable csi-ceph
```

2. Дождитесь перехода модуля в состояние `Ready`:

```
d8 k get module csi-ceph -w
```

3. Убедитесь, что все поды в неймспейсе `d8-csi-ceph` находятся в состоянии `Running` или `Completed` и развёрнуты на всех узлах кластера:

```
d8 k -n d8-csi-ceph get pod -owide -w
```

4. Для настройки подключения к Ceph-кластеру примените ресурс `CephClusterConnection`.

Пример команды:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephClusterConnection
metadata:
  name: ceph-cluster-1
spec:
  # FSID/UUID Ceph-кластера.
  # Получить FSID/UUID Ceph-кластера можно с помощью команды `ceph fsid`.
  clusterID: 2bf085fc-5119-404f-bb19-820ca6a1b07e
  # Список IP-адресов ceph-мон в формате 10.0.0.10:6789.
  monitors:
    - 10.0.0.10:6789
  # Имя пользователя без `client.`.
  # Получить имя пользователя можно с помощью команды `ceph auth list`.
  userID: admin
  # Ключ авторизации, соответствующий userID.
  # Получить ключ авторизации можно с помощью команды `ceph auth get-key
  client.admin`.
  userKey: AQDiVXVmBJVRLxAAg65Ph0DrtwbwSWrjJwssUg==
EOF
```

5. Проверьте создание подключения командой (`Phase` должна быть в статусе `Created`):

```
d8 k get cephclusterconnection ceph-cluster-1
```

6. Создайте объект `StorageClass` при помощи ресурса `CephStorageClass`. Ручное создание `StorageClass` без использования `CephStorageClass` может привести к ошибкам.

Пример создания `StorageClass` на основе RBD:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-rbd-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: RBD
  rbd:
    defaultFSType: ext4
    pool: ceph-rbd-pool
EOF
```

Пример создания `StorageClass` на основе файловой системы Ceph:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-fs-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: CephFS
  cephFS:
    fsName: cephfs
EOF
```

7. Проверьте, что созданные ресурсы `CephStorageClass` перешли в состояние `Created`:

```
d8 k get cephstorageclass
```

В результате будет выведена информация о созданных ресурсах CephStorageClass:

NAME	PHASE	AGE
ceph-rbd-sc	Created	1h
ceph-fs-sc	Created	1h

8. Проверьте созданный StorageClass:

```
d8 k get sc
```

В результате будет выведена информация о созданном StorageClass:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE
ceph-rbd-sc	rbd.csi.ceph.com	Delete	WaitForFirstConsumer
ceph-fs-sc	rbd.csi.ceph.com	Delete	WaitForFirstConsumer

Настройка подключения к Ceph-кластеру завершена. Вы можете использовать созданный StorageClass для создания PersistentVolumeClaim в ваших приложениях.

Дополнительная информация

Получение списка томов RBD, разделённого по узлам

Для мониторинга и диагностики полезно знать, какие RBD-тома подключены к каждому узлу кластера. Следующая команда позволяет получить детальную информацию о маппинге томов:

```
d8 k -n d8-csi-ceph get po -l app=csi-node-rbd -o custom-
columns=NAME:.metadata.name,NODE:.spec.nodeName --no-headers \
| awk '{print "echo \"$2\"; kubectl -n d8-csi-ceph exec \"$1\" -c node -- rbd
showmapped"}' | bash
```

Поддерживаемые версии Ceph-кластеров

Модуль `csi-ceph` предъявляет требования к версии Ceph-кластера, чтобы обеспечить совместимость и стабильную работу. Официально поддерживаются версии Ceph $\geq 16.2.0$. На практике текущая версия модуля обычно работает и с кластерами $\geq 14.2.0$, но для надёжной эксплуатации рекомендуется обновить Ceph до актуальной поддерживаемой версии.

Поддерживаемые режимы работы томов

Различные типы хранилища Ceph поддерживают разные режимы доступа к томам, что важно учитывать при планировании архитектуры приложений.

- **RBD** — поддерживает только ReadWriteOnce (RWO) — доступ к тому только с одного узла кластера.
- **CephFS** — поддерживает ReadWriteOnce (RWO) и ReadWriteMany (RWX) — одновременный доступ к тому с нескольких узлов кластера.

Разрешения (caps) для пользователей в Ceph

Для обеспечения корректной работы модуля `csi-ceph` пользователи Ceph должны иметь соответствующие разрешения (caps). Необходимые разрешения зависят от используемого типа хранилища. Ниже приведены примеры правильных конфигураций разрешений для различных сценариев.

RBD

Для одного пула с названием `rbd` требуются следующие разрешения:


```
[client.name]
  key = key
  caps mgr = "profile rbd pool=rbd"
  caps mon = "profile rbd"
  caps osd = "profile rbd pool=rbd"
```

CephFS

Перед настройкой разрешений CephFS убедитесь, что в CephFS создан subvolumegroup `csi` (или другой, указанный в `Custom resources`).

Создать новый subvolumegroup можно командой на узле управления Ceph:

```
ceph fs subvolumegroup create <fs_name> <group_name>
```

Например, для создания subvolumegroup `csi` для файловой системы `myfs`:

```
ceph fs subvolumegroup create myfs csi
```

Требуемые разрешения для CephFS с названием `myfs`:

```
[client.name]
  key = key
  caps mds = "allow rwps fsname=myfs"
  caps mgr = "allow rw"
  caps mon = "allow r fsname=myfs"
  caps osd = "allow rw tag cephfs data=myfs, allow rw tag cephfs metadata=myfs"
```

CephFS + RBD

Для пользователя, которому необходим доступ к CephFS `myfs` и RBD пулу `rbd`, объедините разрешения следующим образом:

```
[client.name]
  key = key
  caps mds = "allow rwps fsname=myfs"
  caps mgr = "allow rw,profile rbd pool=rbd"
  caps mon = "allow r fsname=myfs,profile rbd"
  caps osd = "allow rw tag cephfs metadata=myfs, allow rw tag cephfs
data=myfs,profile rbd pool=rbd"
```

Дополнительные ресурсы

- Кастомные ресурсы модуля `csi-ceph`
- Документация модуля `snapshot-controller`

Хранилище данных HPE

В Deckhouse реализована поддержка систем хранения данных (СХД) HPE ZPAR для управления томами в Kubernetes с использованием CSI-драйвера. Такая интеграция обеспечивает надежное, масштабируемое и высокопроизводительное хранилище, подходящее для критически важных рабочих нагрузок. Для работы с системами хранения HPE ZPAR применяется модуль `csi-hpe`, с помощью которого можно создавать StorageClass в Kubernetes через создание ресурса HPEStorageClass.



Создание StorageClass для CSI-драйвера `csi.hpe.com` пользователем запрещено. Модулем поддерживаются только СХД HPE ZPAR. Для использования других СХД HPE свяжитесь с технической поддержкой Deckhouse.

На этой странице представлены инструкции по подключению HPE ZPAR в Deckhouse, настройке соединения, созданию StorageClass, а также проверке работоспособности хранилища.

Системные требования

- Наличие развернутой и настроенной СХД HPE.
- Уникальные IQN в `/etc/iscsi/initiatorname.iscsi` на каждом узле Kubernetes.

Настройка

Все команды следует выполнять на машине, имеющей доступ к API Kubernetes с правами администратора.

Включение модуля

Включите модуль `csi-hpe`. Это приведет к тому, что на всех узлах кластера будет:

- зарегистрирован CSI-драйвер;
- запущены служебные поды компонентов `csi-hpe`.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-hpe
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, когда модуль перейдет в состояние `Ready`. Проверьте состояние модуля следующей командой:

```
d8 k get module csi-hpe -W
```

Создание StorageClass

Для создания StorageClass необходимо использовать ресурсы HPEStorageClass и HPEStorageConnection. Пример команд для создания таких ресурсов:

- Создание ресурса HPEStorageConnection:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HPEStorageConnection
metadata:
  name: hpe
spec:
  controlPlane:
    backendAddress: "172.17.1.55" # Адрес СХД (изменяемый параметр).
    username: "Зпарadm" # Имя пользователя для доступа к API (изменяемый параметр).
    password: "Зпарdata" # Пароль для доступа к API (изменяемый параметр).
    serviceName: "primeraЗпар-csp-svc"
    servicePort: "8080"
EOF
```

Проверьте создание объекта следующей командой (`Phase` должен быть `Created`):

```
d8 k get hpestorageconnections.storage.deckhouse.io <имя hpestorageconnection>
```

- Создание ресурса HPEStorageClass:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HPEStorageClass
metadata:
  name: hpe
spec:
  pool: "test-cpg"
  accessProtocol: "fc" # fc или iscsi (по умолчанию iscsi), неизменяемый
  параметр.
  fsType: "xfs" # xfs, ext3, ext4, btrfs (по умолчанию ext4), изменяемый
  параметр.
  storageConnectionName: "3par" # Неизменяемый параметр.
  reclaimPolicy: Delete # Delete или Retain.
  cpg: "test-cpg"
EOF
```

Проверьте создание объекта следующей командой (Phase должен быть Created):

```
d8 k get hpestorageclasses.storage.deckhouse.io <имя hpestorageclass>
```

Проверка работоспособности модуля

Для проверки работоспособности модуля убедитесь, что все поды в пространстве имён `d8-csi-hpe` находятся в статусе `Running` или `Completed` и запущены на каждом узле кластера:

```
d8 k -n d8-csi-hpe get pod -owide -w
```

Дополнительные ресурсы

- Документация модуля `csi-hpe`

Хранилище данных Huawei

В Deckhouse предусмотрена поддержка систем хранения данных (СХД) Huawei Dorado, позволяющая управлять томами в Kubernetes с использованием CSI-драйвера через создание пользовательских ресурсов `HuaweiStorageClass`. Это решение обеспечивает высокую производительность и отказоустойчивость хранения данных, что делает его оптимальным выбором для критически важных рабочих нагрузок.



Создание `StorageClass` для CSI-драйвера `csi.huawei.com` пользователем запрещено. Модулем поддерживаются только СХД Huawei Dorado. Для использования других СХД Huawei свяжитесь с технической поддержкой Deckhouse.

На этой странице представлены инструкции по подключению Huawei Dorado к Deckhouse, настройке соединения, созданию `StorageClass`, а также проверке работоспособности хранилища.

Системные требования

- Наличие развернутой и настроенной СХД Huawei.
- Уникальные IQN в `/etc/iscsi/initiatorname.iscsi` на каждом узле Kubernetes.

Настройка

Все команды следует выполнять на машине, имеющей доступ к API Kubernetes с правами администратора.

Включение модуля

Для поддержки систем хранения данных Huawei Dorado включите модуль `csi-huawei`. Это приведет к тому, что на всех узлах кластера будет:

- зарегистрирован CSI-драйвер;
- запущены служебные поды компонентов `csi-huawei`.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-huawei
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, когда модуль перейдет в состояние `Ready`. Проверьте состояние модуля следующей командой:

```
d8 k get module csi-huawei -w
```

Создание StorageClass

Для создания StorageClass необходимо использовать ресурсы HuaweiStorageClass и HuaweiStorageConnection. Пример команд для создания таких ресурсов:

- Создание ресурса HuaweiStorageConnection:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageConnection
metadata:
  name: huaweistorageconn
spec:
  storageType: OceanStorSAN
  pools:
    - test
  urls:
    - https://192.168.128.101:8088
  login: "admin"
  password: "ivkerg43grdsf_"
  protocol: ISCSI
  portals:
    - 10.240.0.101
    - 10.250.0.101
  maxClientThreads: 30
EOF
```

Проверьте создание объекта следующей командой (Phase должен быть Created):

```
d8 k get huaweistorageconnections.storage.deckhouse.io <имя huaweistorageconnection>
```

- Создание ресурса HuaweiStorageClass:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageClass
metadata:
  name: huaweisc
spec:
  fsType: ext4
  pool: test
  reclaimPolicy: Delete
  storageConnectionName: huaweistorageconn
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Проверьте создание объекта следующей командой (`Phase` должен быть `Created`):

```
d8 k get huaweistorageclasses.storage.deckhouse.io <имя huaweistorageclass>
```

Проверка работоспособности модуля

Для проверки работоспособности модуля убедитесь, что все поды в пространстве имён `d8-csi-huawei` находятся в статусе `Running` или `Completed` и запущены на каждом узле кластера:

```
d8 k -n d8-csi-huawei get pod -owide -w
```

Дополнительные ресурсы

- Документация модуля `csi-huawei`

Хранилище данных NetApp

В Deckhouse Kubernetes Platform (DKP) реализована поддержка систем хранения данных (СХД) NetApp для управления томами в Kubernetes с использованием CSI-драйвера. Такая интеграция обеспечивает надежное, масштабируемое и высокопроизводительное хранилище, подходящее для критически важных рабочих нагрузок. Для работы с системами хранения NetApp применяется модуль `csi-netapp`, с помощью которого можно создавать `StorageClass` в Kubernetes через создание ресурса `NetappStorageClass`.



Создание `StorageClass` для CSI-драйвера `csi.netapp.com` пользователем запрещено. На данный момент модулем поддерживаются СХД, совместимые с Trident CSI от NetApp. Для поддержки других СХД NetApp свяжитесь с технической поддержкой Deckhouse.

На этой странице представлены инструкции по подключению NetApp к DKP, настройке соединения и созданию `StorageClass`.

Системные требования

Перед настройкой работы с СХД NetApp убедитесь, что выполнены следующие требования:

- Наличие развернутой и настроенной СХД NetApp.
- Уникальные IQN в `/etc/iscsi/initiatorname.iscsi` на каждом узле Kubernetes.

Настройка интеграции кластера с системой хранения NetApp

Чтобы начать работу с СХД NetApp, следуйте пошаговым инструкциям ниже. Все команды выполняйте на машине с административным доступом к API Kubernetes.



Для работы со снимками требуется подключенный модуль `snapshot-controller`.

1. Выполните команду для активации модуля `csi-netapp`:

```
d8 system module enable csi-netapp
```

После активации модуля на всех узлах кластера будут:

- Зарегистрирован CSI-драйвер;
- Развернуты служебные поды компонентов `csi-netapp`.

2. Дождитесь перехода модуля в состояние `Ready`:

```
d8 k get module csi-netapp -w
```

3. Убедитесь, что все поды в пространстве имен `d8-csi-netapp` находятся в состоянии `Running` или `Completed` и развернуты на всех узлах кластера:

```
d8 k -n d8-csi-netapp get pod -owide -w
```

4. Создайте ресурс NetappStorageConnection:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NetappStorageConnection
metadata:
  name: netapp
spec:
  controlPlane:
    address: "172.17.1.55"
    username: "admin"
    password: "password"
    svm: "svm1"
EOF
```

5. Проверьте создание объекта следующей командой (Phase должен быть Created):

```
d8 k get netappstorageconnections.storage.deckhouse.io <имя netappstorageconnection>
```

6. Создайте ресурс NetappStorageClass:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NetappStorageClass
metadata:
  name: netapp
spec:
  pool: "test-cpg"
  accessProtocol: "iscsi" # fc или iscsi (по умолчанию iscsi), неизменяемый
  параметр.
  fsType: "xfs" # xfs, ext3, ext4 (по умолчанию ext4), изменяемый параметр.
  storageConnectionName: "netapp" # Неизменяемый параметр.
  reclaimPolicy: Delete # Delete или Retain.
  cpg: "test-cpg"
EOF
```

7. Проверьте создание объекта следующей командой (Phase должен быть Created):

```
d8 k get netappstorageclasses.storage.deckhouse.io <имя netappstorageclass>
```

Теперь система хранения NetApp готова к работе. Вы можете использовать созданный StorageClass для создания PersistentVolumeClaim в ваших приложениях.

Дополнительные ресурсы

- Документация модуля `csi-netapp`
- Документация модуля `snapshot-controller`

Сетевое файловое хранилище NFS

Deckhouse Kubernetes Platform (DKP) поддерживает интеграцию с Network File System (NFS), обеспечивая возможность использования сетевых файловых хранилищ в качестве томов Kubernetes. Модуль `csi-nfs` предоставляет CSI-драйвер для подключения NFS-серверов и создания PersistentVolume на их основе.

На этой странице представлены инструкции по настройке NFS-хранилища в DKP, включая подключение к NFS-серверу, создание StorageClass, настройку безопасности RPC-with-TLS, а также проверку работоспособности системы.

Системные требования

Минимальные требования

Для работы модуля `csi-nfs` необходимо выполнение следующих условий:

- Поддерживаемые дистрибутивы Linux с соответствующими ядрами;
- Настроенный и доступный NFS-сервер;
- Для поддержки RPC-with-TLS: ядро Linux с включенными опциями `CONFIG_TLS` и `CONFIG_NET_HANDSHAKE`.

Рекомендации по настройке

Для оптимальной работы модуля рекомендуется:

- Для автоматического перезапуска подов при изменении параметров TLS включите модуль `pod-reloader` (включен по умолчанию).
- Используйте стабильные версии NFS-сервера с поддержкой необходимых протоколов.



Для работы NFS как хранилища виртуальных дисков в Deckhouse Virtualization Platform настройте NFS-сервер с опцией `no_root_squash`.

Ограничения

Общие ограничения

При работе с модулем `csi-nfs` действуют следующие ограничения:

- Создание StorageClass для CSI-драйвера `nfs.csi.k8s.io` вручную запрещено — используйте ресурс `NFSStorageClass`.
- PersistentVolume создаются только через `NFSStorageClass` ресурсы.
- Изменение параметров NFS-сервера в уже созданных PV невозможно.

Ограничения снимков томов



Для работы со снимками требуется подключенный модуль `snapshot-controller`.

Создание снимков NFS-томов имеет существенные ограничения, связанные с архитектурой NFS и способом их реализации в `csi-nfs`. По возможности избегайте использования снимков в модуле.

Принцип работы снимков:

- CSI-драйвер создает снимок на уровне NFS-сервера
- Для архивирования используется утилита `tar`, что накладывает определенные ограничения
- Архив сохраняется в корневой папке NFS-сервера, указанной в параметре `spec.connection.share`.



- Перед созданием снимка **обязательно** остановите рабочую нагрузку (поды), использующую NFS-том.
- NFS не обеспечивает атомарность операций на уровне файловой системы при создании снимка.

Ограничения режима RPC-with-TLS

Функциональные ограничения

При использовании RPC-with-TLS действуют следующие ограничения:

- Для политики безопасности `mtls` поддерживается только один сертификат клиента.
- Один NFS-сервер не может одновременно работать в разных режимах безопасности: `tls`, `mtls` и стандартный режим (без TLS).
- На узлах кластера не должен быть запущен демон `tlshd`, иначе он будет конфликтовать с демоном модуля. Для предотвращения конфликтов при включении TLS на узлах автоматически останавливается сторонний `tlshd` и отключается его автозапуск.

Требования к системе

Для работы RPC-with-TLS необходимо выполнение следующих системных требований:

- Ядро Linux должно быть собрано с включенными параметрами `CONFIG_TLS` и `CONFIG_NET_HANDSHAKE`.
- Пакет `nfs-utils` (в дистрибутивах на основе Debian — `nfs-common`) версии `>= 2.6.3`.

Настройка

i Все команды должны выполняться на машине с административными правами в Kubernetes API.

Для настройки NFS-хранилища необходимо выполнить следующие шаги:

- Включение модуля;
- Создание NFSStorageClass.

Включение модуля

Для поддержки работы с NFS-хранилищем включите модуль `csi-nfs`, который позволяет создавать StorageClass в Kubernetes с помощью пользовательских ресурсов NFSStorageClass. После включения модуля на узлах кластера произойдёт:

- Регистрация CSI-драйвера;
- Запуск сервисных подов `csi-nfs` и создание необходимых компонентов.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-nfs
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, пока модуль перейдет в состояние `Ready`:

```
d8 k get module csi-nfs -w
```

Проверьте состояние подов в пространстве имён `d8-csi-nfs`. Все поды должны быть в состоянии `Running` или `Completed`, и запущены на всех узлах:

```
d8 k -n d8-csi-nfs get pod -owide -w
```

Создание StorageClass

Для создания StorageClass необходимо использовать ресурс NFSStorageClass. Пример создания ресурса:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NFSStorageClass
metadata:
  name: nfs-storage-class
spec:
  connection:
    host: 10.223.187.3
    share: /
    nfsVersion: "4.1"
    reclaimPolicy: Delete
    volumeBindingMode: WaitForFirstConsumer
EOF
```

Для каждого PV будет создаваться каталог `<директория из share>/<имя PV>`.

Настройка очистки томов

При удалении PersistentVolume на томе могут остаться файлы с пользовательскими данными. Для обеспечения безопасности данных можно настроить метод очистки тома перед удалением с помощью параметра `volumeCleanup`.



Важные замечания по очистке томов:

- Опция очистки не влияет на файлы, уже удаленные клиентским приложением.
- Очистка выполняется только через протокол NFS и зависит от:
 - Сервиса NFS-сервера;
 - Файловой системы;
 - Уровня блочных устройств и их виртуализации (например, LVM);
 - Физических устройств.
- Убедитесь в доверенности NFS-сервера перед отправкой чувствительных данных.

Параметр `RandomFillSinglePass`

Содержимое файлов переписывается случайной последовательностью перед удалением. Случайная последовательность передается по сети.

Параметр `RandomFillThreePass`

Содержимое файлов трижды переписывается случайной последовательностью перед удалением. Три случайных последовательности передаются по сети.



Использование данного метода имеет смысл только в том случае, если сервер хранит данные на жестком диске, и есть риск физического доступа злоумышленника к устройству.

Параметр `Discard`

Оптимизированный метод очистки, использующий возможности файловых систем для работы с твердотельными накопителями. Содержимое файлов помечается как свободное через системный вызов `falloc` с флагом `FALLOC_FL_PUNCH_HOLE`. Файловая система освободит полностью используемые файлом блоки через вызов `blkdiscard`, а остальное место будет перезаписано нулями.

Преимущества метода `Discard`:

- Объём трафика не зависит от размера файлов, а только от их количества;
- Может обеспечить недоступность старых данных при некоторых конфигурациях сервера;
- Работает как для жестких дисков, так и для твердотельных накопителей;
- Позволяет увеличить время жизни твердотельного накопителя.

Изменение параметров NFS-сервера для созданных PersistentVolume

Изменение параметров NFS-сервера для созданных PersistentVolume невозможно, так как данные для подключения к NFS-серверу сохраняются непосредственно в манифесте PersistentVolume и не подлежат изменению. Изменение NFSStorageClass также не повлечет изменений настроек подключения в уже существующих PV.

Создание снимков томов

В `csi-nfs` снимки создаются путем архивирования папки тома. Архив сохраняется в корне папки NFS-сервера, указанной в параметре `spec.connection.share`. При создании снимка томов важно учитывать требования и ограничения, перечисленные выше.

Чтобы создать снимок, выполните следующие действия:

1. Включите модуль `snapshot-controller`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: snapshot-controller
spec:
  enabled: true
  version: 1
EOF
```

2. Создайте снимок тома, указав нужные параметры:

```
d8 k apply -f - <<EOF
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: my-snapshot
  namespace: <имя namespace, в котором находится PVC>
spec:
  volumeSnapshotClassName: csi-nfs-snapshot-class
  source:
    persistentVolumeClaimName: <имя PVC, для которого необходимо создать
снимок>
EOF
```

3. Проверьте состояние созданного снимка:

```
d8 k get volumesnapshot
```

Проблемы с удалением PV с поддержкой RPC-with-TLS

Если ресурс NFSStorageClass был настроен с поддержкой RPC-with-TLS, может возникнуть ситуация, когда PersistentVolume не удастся удалить. Это происходит из-за удаления секрета (например, после удаления NFSStorageClass, который хранит параметры монтирования. В результате контроллер не может смонтировать NFS-папку для удаления папки <имя PV>.

Добавление нескольких CA в параметр `tlsParameters.ca`

Для добавления нескольких сертификатов CA в параметр `tlsParameters.ca` используйте следующие команды:

Для двух CA:

```
cat CA1.crt CA2.crt | base64 -w0
```

Для трех CA:

```
cat CA1.crt CA2.crt CA3.crt | base64 -w0
```

Для большего количества CA:

```
cat CA1.crt CA2.crt CA3.crt ... CAN.crt | base64 -w0
```

Дополнительные ресурсы

- Документация модуля `csi-nfs`
- Документация модуля `pod-reloader`
- Документация модуля `snapshot-controller`

Объектное хранилище на основе S3

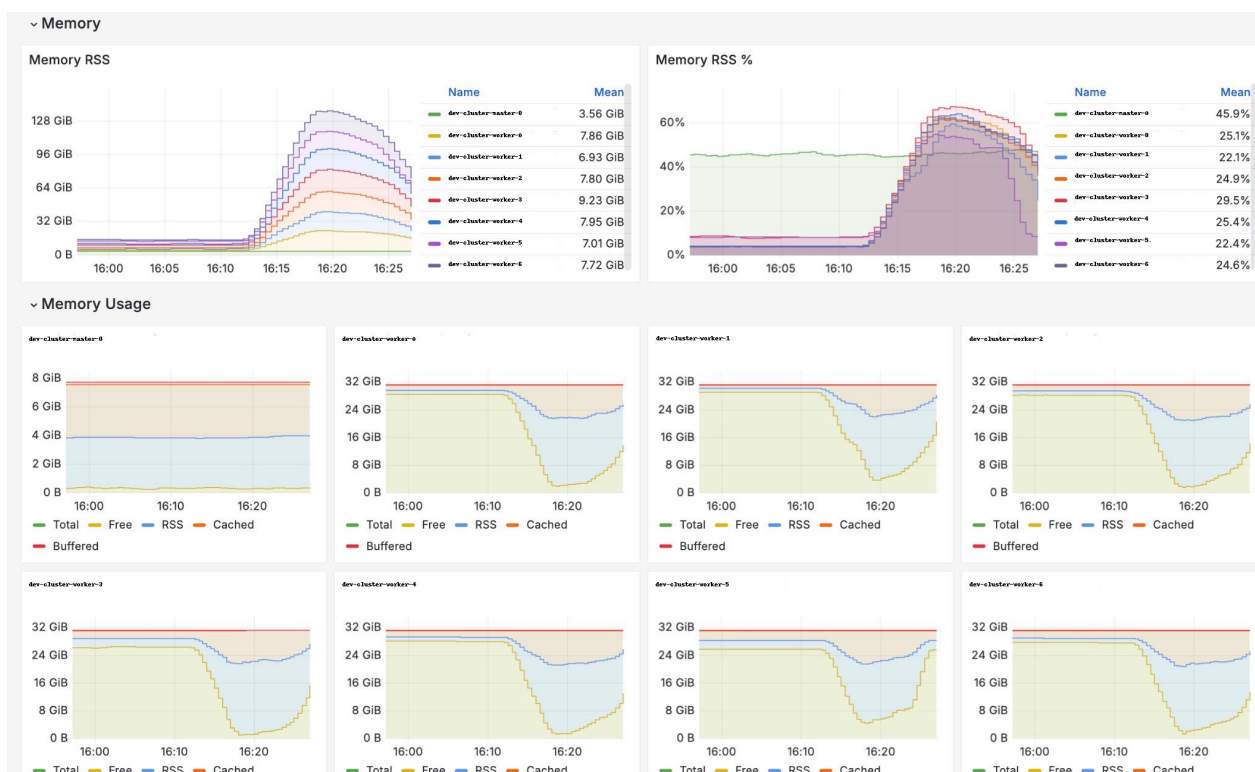
Deckhouse поддерживает работу с объектным хранилищем на основе S3, обеспечивая

возможность использования его в Kubernetes для хранения данных в виде томов. В качестве файловой системы применяется GeeseFS, работающая через FUSE поверх S3, что позволяет монтировать S3-хранилища как стандартные файловые системы.

На этой странице представлены инструкции по настройке S3-хранилища в Deckhouse, включая подключение, создание StorageClass, а также проверку работоспособности системы.

Системные требования

- Kubernetes версии 1.17+ с поддержкой привилегированных контейнеров.
- Настроенное S3-хранилище с доступными ключами.
- Достаточный объем памяти на узлах. GeeseFS использует кеш для работы с файлами, загружаемыми из S3. Размер кеша задается параметром `maxCacheSize` в `S3StorageClass`. Результаты стресс-теста: 7 узлов, 600 подов и PVC, `maxCacheSize` = 500 МБ, каждый под записывает 300 МБ, читает их и завершает работу.



Настройка

Обратите внимание, что все команды должны выполняться на машине с административными правами в Kubernetes API.

Необходимые шаги:

- включение модуля;
- создание `S3StorageClass`.

Включение модуля

Для поддержки работы с S3-хранилищем включите модуль `csi-s3`, который позволяет создавать `StorageClass` и `Secret` в Kubernetes с помощью пользовательских ресурсов `S3StorageClass`. После включения модуля на узлах кластера произойдет:

- регистрация CSI-драйвера;
- запуск сервисных подов `csi-s3` и создание необходимых компонентов.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-s3
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, пока модуль не перейдет в состояние `Ready`.

```
d8 k get module csi-s3 -w
```

Создание StorageClass

Модуль настраивается с помощью манифеста `S3StorageClass`. Пример конфигурации:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: S3StorageClass
metadata:
  name: example-s3
spec:
  bucketName: example-bucket
  endpoint: https://s3.example.com
  region: us-east-1
  accessKey: <your-access-key>
  secretKey: <your-secret-key>
  maxCacheSize: 500
  insecure: false
```

Если `bucketName` не указан, для каждого `PersistentVolume` будет создан отдельный bucket. Если `bucketName` задан, в нем будут создаваться папки под каждый `PersistentVolume`. Если такого bucket нет, он будет создан автоматически.

Проверка работоспособности модуля

Для проверки работоспособности модуля убедитесь, что все поды в пространстве имён `d8-csi-s3` находятся в статусе `Running` или `Completed` и запущены на каждом узле кластера:

```
d8 k -n d8-csi-s3 get pod -owide -w
```

Ограничения

S3 не является полноценной файловой системой, поэтому существуют определенные ограничения. Совместимость с POSIX зависит от используемого монтирующего модуля и провайдера S3. Некоторые хранилища могут не гарантировать консистентность данных (подробнее).

Обратите внимание на таблицу совместимости с POSIX.

Основные ограничения:

- Разрешения файловой системы, символические ссылки, пользовательские `mtimes` и специальные файлы (блочные/символьные устройства, именованные каналы, сокеты UNIX) не поддерживаются, поскольку стандарт S3 не возвращает метаданные пользователя в списках и не читает все эти метаданные в стандарте S3. Потребуется дополнительный запрос `HEAD` для каждого файла в листинге, что сделает листинги слишком медленными.
- Поддержка специальных файлов включена по умолчанию для `Яндекс S3` и отключена для других.
- Разрешения файловой системы отключены по умолчанию.
- Пользовательское время модификации также отключено по умолчанию: `ctime`, `atime` и `mtime` всегда одинаковы.
- Время изменения файла не может быть установлено пользователем (например, с помощью

```
cp --preserve, rsync -a или utimes(2)
```

- Не поддерживаются жесткие ссылки.
- Не поддерживается блокировка.
- Не поддерживаются «невидимые» удаленные файлы. Если приложение сохраняет открытый файловый дескриптор после удаления файла, оно получит ошибки `ENOENT` от операций FS.
- Ограничение размера файла по умолчанию составляет 1,03 ТБ и достигается путем разделения файла на 1000 частей по 5 МБ, 1000 частей по 25 МБ и 8000 частей по 125 МБ. Вы можете изменить размеры частей, но собственный лимит AWS в любом случае составляет 5 ТБ.

Известные ошибки и проблемы

- Запрос размера тома в PVC никак не отражается на создаваемых bucket.
- `df -h` показывает размер смонтированного хранилища в 1 ПБ, `used` никак не меняется во время использования.
- CSI не проверяет корректность ключей для доступа к хранилищу, статус пода будет `Running`, а `PersistentVolume` и `PersistentVolumeClaim` будут `Bound` даже если ключи неправильные. Попытка доступа к смонтированной директории в поде приведёт к перезапуску пода.

Изменение параметров подключения для существующих PV

Параметры подключения к хранилищу изменить нельзя. Изменение `StorageClass` не обновляет настройки подключения в уже созданных PV.

Особенность отображения размера монтированной директории

Команда `df -h` показывает размер смонтированного тома как 1 ПБ, а значение `Used` остаётся неизменным в процессе работы. Это особенность модуля монтирования `GeeseFS`.

Обработка превышения квоты bucket или пользователя

Превышение квоты — нештатная ситуация, которой следует избегать. Поведение при этом зависит от используемого бэкенда:

- файлы можно копировать и редактировать в подах, но изменения не отражаются в хранилище;
- под может выйти из строя и перезапуститься.

Получение информации об используемом пространстве

Для получения информации об используемом пространстве воспользуйтесь веб-интерфейсом хранилища или командной строкой.

Использование нескольких S3-хранилищ в одном модуле

Чтобы использовать несколько S3-хранилищ, создайте дополнительный `S3StorageClass` и соответствующий PVC, а затем смонтируйте их в поде следующим образом:

```

d8 k apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: csi-s3-test-nginx
  namespace: default
spec:
  containers:
    - name: csi-s3-test-nginx
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html/s3
          name: webroot
        - mountPath: /opt/homedir
          name: homedir
  volumes:
    - name: webroot
      persistentVolumeClaim:
        claimName: csi-s3-pvc # PVC name
        readOnly: false
    - name: homedir
      persistentVolumeClaim:
        claimName: csi-s3-pvc2 # PVC-2 name
        readOnly: false
EOF

```

Использование одного bucket для нескольких подов

При указании параметра `bucketName` в `S3StorageClass` в одном bucket для каждого PV создаются отдельные каталоги, что позволяет использовать один bucket для нескольких подов.

Диагностика неисправностей

Проблемы при создании PVC

При возникновении проблем при создании PVC проверьте логи `provisioner`:

```
d8 k -n d8-csi-s3 logs -l app=csi-provisioner-s3 -C csi-s3
```

Проблемы с запуском контейнеров

Убедитесь, что для `MountPropagation` не установлено значение `false`, и проверьте логи S3-драйвера:

```
d8 k -n d8-csi-s3 logs -l app=csi-s3 -C csi-s3
```

Дополнительные ресурсы

- Документация модуля `csi-s3`

Хранилище данных на основе протокола SCSI

Deckhouse Kubernetes Platform (DKP) поддерживает управление хранилищами, подключёнными через iSCSI или Fibre Channel, обеспечивая возможность работы с томами на уровне блочных устройств. Это позволяет интегрировать системы хранения данных с Kubernetes и управлять ими через CSI-драйвер.

На этой странице представлены инструкции по подключению SCSI-устройств в DKP, созданию `SCSITarget`, `StorageClass`, а также проверке работоспособности системы.

Поддерживаемые возможности

DKP поддерживает:

- обнаружение логических томов хранения данных (Logical Unit Number, LUN) через iSCSI или Fibre Channel;
- создание PersistentVolume (PV) из заранее подготовленных LUN;
- удаление PV и обнуление данных на LUN;
- подключение LUN к узлам через iSCSI или Fibre Channel;
- создание multipath-устройств и их монтирование в поды;
- отключение LUN от узлов.

Ограничения

DKP не поддерживает:

- создание LUN на СХД;
- изменение размера LUN;
- создание снимков (snapshots).

Системные требования

Требования к инфраструктуре и узлам кластера:

- Развёрнутая и настроенная СХД, предоставляющая доступ к LUN по iSCSI или Fibre Channel.
- Для подключения по iSCSI:
 - на каждом узле кластера должен быть настроен уникальный идентификатор iSCSI-инициатора (iSCSI Qualified Name, IQN) в файле `/etc/iscsi/initiatorname.iscsi`;
 - на узлах должен быть установлен пакет `multipath-tools`.
- Для подключения по Fibre Channel:
 - на узлах кластера должны быть установлены и доступны адаптеры Fibre Channel (Fibre Channel Host Bus Adapter, FC HBA) (`/sys/class/fc_host/host*`);
 - в сети хранения данных (Storage Area Network, SAN) должны быть настроены зонирование и маскирование LUN, обеспечивающие доступ инициаторов узлов к портам СХД;
 - необходимые LUN должны быть заранее созданы в СХД. DKP не создаёт LUN;
 - на узлах должен быть установлен пакет `multipath-tools`.

Быстрый старт

Все команды следует выполнять на машине, имеющей доступ к API Kubernetes с правами администратора.

Включение модуля `csi-scsi-generic`

Включите модуль `csi-scsi-generic` с помощью команды ниже. Это приведет к тому, что на всех узлах кластера будет:

- зарегистрирован CSI-драйвер;
- запущены служебные поды компонентов `csi-scsi-generic`.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-scsi-generic
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, когда модуль перейдет в состояние `Ready`:

```
d8 k get module csi-scsi-generic -w
```

Создание `SCSITarget`

Ресурс `SCSITarget` описывает подключение к одной SCSI-цели. При создании ресурса в `spec` укажите один из способов подключения: `iSCSI` или `fibreChannel`.

iSCSI

Далее приведён пример конфигурации для подключения по iSCSI. В этом примере создаются два ресурса `SCSITarget`.

Можно создать несколько ресурсов как для одной, так и для разных СХД, что позволяет использовать `multipath` для повышения отказоустойчивости и производительности.

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-1
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx1
    portals:
      - 192.168.1.1

---
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-2
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx2
    portals:
      - 192.168.1.2
EOF
```

Проверить создание объекта можно следующей командой:

```
d8 k get scsitargets.storage.deckhouse.io <имя-scsitarget>
```

Объект считается успешно созданным, если в колонке его состояния (`Phase`) в выводе указано `Created`.

Fibre Channel

Для настройки подключения по Fibre Channel выполните следующие шаги:

1. Перед созданием ресурса `SCSITarget` настройте на SAN зонирование и маскирование LUN, чтобы узлы кластера получили доступ к необходимым томам.
2. Создайте ресурс `SCSITarget`. В поле `spec.fibreChannel.wwns` укажите World Wide Port Name (WWPN) портов СХД, через которые доступны необходимые LUN. DKP обнаруживает устройства, сопоставляя указанные WWPN с путями в директории `/dev/disk/by-path/`.

WWPN следует указывать в одном из следующих форматов:

- 16 шестнадцатеричных символов (`2001c89f1acd6117`);

- с двоеточиями (20:01:c8:9f:1a:cd:61:17);
- с префиксом 0x (0x2001c89f1acd6117).

При обнаружении и подключении тома DKP выполняет сканирование FC-хостов. При этом он не настраивает коммутаторы и не выполняет вход на целевые устройства.

Пример конфигурации с двумя портами СХД для организации multipath:

```
```shell d8 k apply -f -<EOF apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSTarget metadata:
name: hpe-3par-fc-1 spec: deviceTemplate: metadata: labels: my-key: some-label-value
fibreChannel: WWNs: - 2001c89f1acd6117
```

```
apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSTarget metadata: name: hpe-3par-fc-2 spec:
deviceTemplate: metadata: labels: my-key: some-label-value fibreChannel: WWNs: -
2001c89f1acd6118 EOF
```

1. Перед применением ресурса убедитесь, что на узле доступны адаптеры FC HBA и что после настройки зонирования на SAN видны пути к LUN:

```
```shell
# FC-хосты присутствуют.
ls /sys/class/fc_host/

# WWPN целевых устройств и LUN доступны после настройки зонирования.
ls -l /dev/disk/by-path/ | grep -E 'fc-|/fc-'
```

Результаты проверки:

- Если директория `/sys/class/fc_host/` пуста, на узле отсутствуют FC HBA или не загружен драйвер для них.
 - Если в директории `/dev/disk/by-path/` отсутствуют записи вида `fc-*`, проверьте настройки зонирования и маскирование LUN на СХД.
 - Если FC-пути отображаются, узел готов обнаруживать LUN, предоставленные через Fibre Channel.
1. После создания SCSTarget контроллер обнаружит доступные LUN и создаст объекты SCSIDevice. В дальнейшем используйте тот же ресурс SCSIStorageClass и селектор `scsiDeviceSelector`, что и для iSCSI.
 2. Проверьте создание объекта следующей командой:

```
d8 k get scsitargets.storage.deckhouse.io <имя-scsitarget>
```

Объект считается успешно созданным, если в колонке его состояния (`Phase`) в выводе указано `Created`.

Создание StorageClass

Для создания StorageClass используйте ресурс SCSIStorageClass. Пример команды для создания такого ресурса:

```
d8 k apply -f -<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSIStorageClass
metadata:
  name: scsi-all
spec:
  scsiDeviceSelector:
    matchLabels:
      my-key: some-label-value
  reclaimPolicy: Delete
EOF
```

Параметр `scsiDeviceSelector` позволяет выбрать SCSTarget для создания PV по лейблам. В примере выше выбираются все SCSTarget с лейблом `my-key: some-label-value`. Этот лейбл будет назначен на все устройства, которые будут обнаружены в указанных SCSTarget.

Проверить создание объекта можно следующей командой:

```
d8 k get scsistorageclasses.storage.deckhouse.io <имя-scsistorageclass>
```

Объект считается успешно созданным, если в колонке его состояния (`Phase`) в выводе указано `Created`.

Проверка работоспособности модуля `csi-scsi-generic`

Проверьте состояние подов в неймспейсе `d8-csi-scsi-generic` при помощи команды ниже. Все поды должны быть в состоянии `Running` или `Completed` и запущены на всех узлах.

```
d8 k -n d8-csi-scsi-generic get pod -owide -w
```

Дополнительные ресурсы

- Документация модуля `csi-scsi-generic`

Унифицированное хранилище TATLIN.UNIFIED (Yadro)

Deckhouse поддерживает интеграцию с системой хранения данных TATLIN.UNIFIED (Yadro), предоставляя возможность управления томами в Kubernetes. Это позволяет использовать централизованное хранилище для контейнеризированных рабочих нагрузок, обеспечивая высокую производительность и отказоустойчивость.

На этой странице представлены инструкции по подключению TATLIN.UNIFIED (Yadro) к Deckhouse, настройке соединения, созданию StorageClass, а также проверке работоспособности системы.

Включение модуля

Для управления томами на основе системы хранения данных TATLIN.UNIFIED (Yadro) в Deckhouse используется модуль `csi-yadro-tatlin-unified`, позволяющий создавать ресурсы StorageClass через создание пользовательских ресурсов YadroTatlinUnifiedStorageClass. Чтобы включить модуль, выполните команду:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-yadro-tatlin-unified
spec:
  enabled: true
  version: 1
EOF
```

Дождитесь, когда модуль `csi-yadro-tatlin-unified` перейдет в состояние `Ready`. Проверить состояние модуля можно, выполнив следующую команду:

```
d8 k get module csi-yadro-tatlin-unified -w
```

В результате будет выведена информация о модуле:

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
si-yadro-tatlin-unified		Embedded	Available	True	True

Подключение к системе хранения данных TATLIN.UNIFIED

Чтобы создать подключение к системе хранения данных TATLIN.UNIFIED и иметь возможность настраивать объекты StorageClass, примените следующий ресурс YadroTatlinUnifiedStorageConnection:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageConnection
metadata:
  name: yad1
spec:
  controlPlane:
    address: "172.19.28.184"
    username: "admin"
    password: "cGFzc3dvcmQ=" # Должен быть закодирован в Base64.
    ca: "base64encoded"
    skipCertificateValidation: true
  dataPlane:
    protocol: "iscsi"
    iscsi:
      volumeExportPort: "p50,p51,p60,p61"
EOF
```

Создание StorageClass

Для создания StorageClass необходимо использовать ресурс YadroTatlinUnifiedStorageClass. Ручное создание ресурса StorageClass без YadroTatlinUnifiedStorageClass может привести к ошибкам.

Пример команды для создания класса хранения на основе системы хранения данных TATLIN.UNIFIED:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageClass
metadata:
  name: yad1
spec:
  fsType: "xfs"
  pool: "pool-hdd"
  storageConnectionName: "yad1"
  reclaimPolicy: Delete
EOF
```

Проверка работоспособности модуля

Для того чтобы проверить работоспособность модуля `csi-yadro-tatlin-unified`, необходимо проверить состояние подов в неймспейсе `d8-csi-yadro-tatlin-unified`. Все поды должны быть в состоянии `Running` или `Completed`, поды `csi-yadro-tatlin-unified` должны быть запущены на всех узлах.

Проверить работоспособность модуля можно с помощью команды:

```
d8 k -n d8-csi-yadro-tatlin-unified get pod -owide -w
```

Дополнительные ресурсы

- Документация модуля `csi-yadro-tatlin-unified`

Безопасность

Раздел «Безопасность» посвящён вопросам обеспечения безопасности в Deckhouse Kubernetes Platform. Здесь собраны рекомендации, инструкции и примеры настройки встроенных механизмов защиты, а также интеграции с внешними системами.

В разделе вы найдёте следующую информацию:

- Аудит событий безопасности:
 - как включить и настроить аудит событий API Kubernetes;

- как собирать события безопасности на уровня ядра и API Kubernetes, используя встроенные возможности платформы (Falco);
- как настраивать правила аудита и получать оповещения о подозрительной активности.
- Политики безопасности:
 - поддержка Pod Security Standards;
 - настройка операционных и расширенных политик безопасности с использованием Gatekeeper;
 - проверка подписей контейнерных образов;
 - работа с собственными политиками и исключениями.
- Сканирование образов на уязвимости:
 - как организовать регулярное сканирование контейнерных образов;
 - как просматривать результаты и вручную запускать повторную проверку.
- Управление сертификатами:
 - выпуск, продление и контроль TLS-сертификатов с помощью модуля `cert-manager`;
 - примеры использования Let's Encrypt, HashiCorp Vault, самоподписанных и внешних CA;
 - поддержка видов валидации HTTP-01 и DNS-01.
- Интеграция с внешними системами мониторинга и безопасности:
 - отправка логов в Kaspersky Unified Monitoring and Analysis Platform (KUMA);
 - настройка исключений для антивирусных решений на примере Kaspersky Endpoint Security for Linux (KESL).

Дополнительные ресурсы

- Документация модуля `cert-manager`

События безопасности

Deckhouse Kubernetes Platform (DKP) предоставляет средства для декларативного сбора, обработки, нормализации и доставки событий безопасности, извлекаемых из логов приложений и инфраструктурных компонентов Kubernetes.

Событие безопасности — это структурированная запись о значимом с точки зрения информационной безопасности (ИБ) действии или факте.

DKP позволяет:

- собирать события безопасности из различных источников (логи подов, файлы на узлах, аудит Kubernetes API);
- приводить события к единому формату с обязательным набором атрибутов;
- обогащать события контекстными данными;
- фильтровать события по источникам и уровню критичности;
- доставлять события в системы хранения и аналитики (Loki, Elasticsearch, Kafka, Splunk, Vector и другие).

За сбор, обработку и доставку событий безопасности отвечает модуль `security-events-manager`. Для сбора логов задействуется вспомогательный модуль `log-shipper`.

Зависимости и требования

Для работы модуля `security-events-manager` требуются следующие модули DKP:

- `log-shipper` — обеспечивает сбор логов из подов и файлов на узлах, а также выполняет предварительный отбор записей;
- `loki` — обеспечивает хранение событий безопасности внутри кластера (используется в качестве приёмника событий по умолчанию).

Подробности об архитектуре работы с событиями безопасности можно найти в разделе «Архитектура событий безопасности».

Источники данных для событий безопасности

Модуль `security-events-manager` собирает данные из двух типов источников:

- **Контейнерные источники** — логи приложений в подах Kubernetes. Сбор выполняется через модуль `log-shipper`, который отбирает записи по лейблам подов и неймспейсам.
- **Кластерные источники** — файлы на узлах и логи системных сервисов, не привязанные к конкретному неймспейсу. Например:
 - `/var/log/kube-audit/audit.log` — лог аудита Kubernetes API;
 - `/var/log/auth.log` — лог системных событий аутентификации узла.

Сбор выполняется в два этапа:

1. Модуль `log-shipper` предварительно отбирает логовые записи с помощью простых операций сравнения (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) и передаёт их в шлюз `security-events-manager`.
2. Шлюз `security-events-manager` выполняет парсинг записей, преобразует их в единую модель и отправляет в настроенный приёмник.

Поскольку парсинг логов — это ресурсоёмкая операция, он выполняется только для записей, заранее отобранных как потенциально содержащие события безопасности.

Включение событий безопасности

Чтобы включить события безопасности, выполните следующие шаги:

1. Включите необходимые модули, если они ещё не включены:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  enabled: true
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  enabled: true
```

2. Включите модуль `security-events-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: security-events-manager
spec:
  enabled: true
```

Все доступные параметры `security-events-manager` перечислены в документации модуля.

Настройка источников событий

Источники событий безопасности настраиваются через следующие кастомные ресурсы:

- `PodSecurityEventShipper` — для контейнерных источников (логи подов в конкретном неймспейсе);
- `ClusterSecurityEventShipper` — для кластерных источников (файлы на узлах, логи системных сервисов).

В этих ресурсах настраиваются:

- источник логов (поля `source` и `input`);
- правила первичного отбора записей (поле `produces`);
- правила парсинга (поле `parser` или `parserRef` со ссылкой на ресурсы `SecurityEventLoggingTransformationRules` или `ClusterSecurityEventLoggingTransformationRules`);
- правила преобразования и обогащения (поля `transform` и `enrich`).

Пример конфигурации `PodSecurityEventShipper` для настройки контейнерного источника:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: PodSecurityEventShipper
metadata:
  name: runtime-audit-engine
  namespace: d8-runtime-audit-engine
spec:
  - source: runtime-audit-engine
    input:
      type: KubernetesPods
      kubernetesPods:
        labelSelector:
          matchLabels:
            app: runtime-audit-engine
    produces:
      - eventCode: K8S_POD_CREATED
        extract:
          field: message
          operator: Regex
          values:
            - ".*K8s Pod Created.*"

```

Пример конфигурации ClusterSecurityEventShipper для настройки кластерного источника:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventShipper
metadata:
  name: kube-audit
spec:
  - source: kube-audit
    input:
      type: File
      files:
        - /var/log/kube-audit/audit.log
    produces:
      - eventCode: UNAUTHORIZED_ACCESS
        extract:
          field: message
          operator: Regex
          values:
            - ".*\"code\":401.*"

```

Настройка отправки событий

После формирования событий необходимо определить, в какие приёмники они будут отправляться. Для этого требуется:

1. Настроить приёмники событий безопасности.
2. Настроить правила отправки событий в приёмники.

Настройка приёмника

Приёмники настраиваются через ресурс ClusterSecurityEventDestination.

Поддерживаются все типы приёмников, доступные в экосистеме `log-shipper` (Loki, Elasticsearch, Kafka, Splunk, Vector, File и другие). Для хранения событий безопасности внутри кластера предусмотрена автоматическая настройка приёмника Loki при включении соответствующей опции модуля.

Пример конфигурации ClusterSecurityEventDestination для настройки приёмника Loki:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventDestination
metadata:
  name: cluster-loki
spec:
  type: Loki
  loki:
    auth:
      strategy: Bearer
      token: <EXAMPLE>
    endpoint: https://loki.d8-monitoring:3100
    tls:
      verifyCertificate: true
      verifyHostname: true
      ca: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...

```

Настройка правил отправки событий

При настройке правил отправки событий безопасности необходимо определить:

- из каких источников будут отправляться события;
- минимальный уровень критичности (severity) для отправки;
- приёмники, в которые будут отправляться события.

Для этого используется ресурс ClusterSecurityEventConfig. В ресурсе указываются источники (в формате точных имён или масок), минимальный уровень критичности (severity) и массив приёмников (ресурсы ClusterSecurityEventDestination), в каждый из которых будут отправляться отобранные события.

Пример конфигурации ClusterSecurityEventConfig для отправки событий из источников kube-audit, runtime-audit-engine и user-authn в приёмник cluster-loki:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventConfig
metadata:
  name: default
spec:
  defaultSeverityThreshold: Low
  enabledSources:
    - clusterSecurityEventShipper/kube-audit/kube-audit
    - podSecurityEventShipper/d8-runtime-audit-engine/runtime-audit-engine/falco
    - podSecurityEventShipper/d8-user-authn/user-authn/dex
  # Либо укажите маски.
  # enabledSourcesMasks:
  #   - clusterSecurityEventShipper/kube-audit/*
  #   - podSecurityEventShipper/*

  destinations:
    - cluster-loki

```

Стандартные настройки модуля

Если настройки модуля не заданы явно, в кластере автоматически создаются следующие объекты:

- ClusterSecurityEventConfig — отвечает за настройку отправки событий безопасности в приёмники. По умолчанию создаётся объект со следующими настройками:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventConfig
metadata:
  name: default
spec:
  defaultSeverityThreshold: Medium
  destinations:
    - cluster-loki
  enabledSourcesMasks:
    - podSecurityEventShipper/*
    - clusterSecurityEventShipper/*

```

Настройка этого объекта выполняется с помощью параметра `securityEventConfig`.

- `ClusterSecurityEventDestination` — отвечает за настройку приёмника событий безопасности. По умолчанию создаётся следующий объект, который отправляет события в сервис Loki, расположенный внутри кластера:

```

apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventDestination
metadata:
  name: cluster-loki
spec:
  type: Loki
  loki:
    auth:
      strategy: Bearer
      token: <token> # Заполняется автоматически.
    endpoint: https://loki.d8-monitoring:3100

```

Отключить создание стандартного приёмника можно с помощью параметра `clusterSecurityEventDestination.clusterLoki`.

Поддерживаемые события безопасности

Модуль `security-events-manager` поставляется со встроенным набором правил обнаружения событий безопасности, охватывающим аутентификацию, конфигурацию, RBAC, среду выполнения и другие категории. Актуальный перечень поддерживаемых событий безопасности, их коды, критичность и описание приведены в документации модуля.

Дополнительные ресурсы

- Документация модуля `log-shipper`
- Документация модуля `loki`
- Документация модуля `security-events-manager`

Аудит событий API Kubernetes

Процедура аудита (Kubernetes auditing) позволяет отслеживать обращения к API-серверу и анализировать события, происходящие в кластере. Аудит может быть полезен для отладки неожиданных сценариев поведения, а также для соблюдения требований безопасности.

Kubernetes поддерживает настройку аудита через механизм Audit policy, который позволяет задавать правила логирования интересующих операций. Результаты аудита по умолчанию записываются в лог-файл `/var/log/kube-audit/audit.log`.

Встроенные политики аудита

В Deckhouse Kubernetes Platform (DKP) по умолчанию разворачивается набор встроенных политик аудита. Они фиксируют следующие категории событий:

Общие события:

- Создание и удаление узлов кластера (объектов `Node`);
- Создание, изменение и удаление ресурсов во всех системных пространствах имён (`kube-system`, `d8-*`);

- Запросы от сервисных аккаунтов из системных пространств имён (kube-system, d8-*);
- Массовые запросы (LIST) ко всем ресурсам (для диагностики высокого потребления ресурсов API-сервером);
- Операции с ресурсами модуля virtualization;
- Действия в пространстве имён d8-virtualization;
- Операции с объектами ModuleConfig;
- Запросы от пользователей, не прошедших аутентификацию (логируются только события уровня Metadata).

События, подлежащие регистрации в соответствии с требованиями ФСТЭК к средствам контейнеризации:

- Создание, изменение и удаление объектов Pod;
- Использование тегов образов контейнеров, отличных от формата @sha256;
- Создание и удаление объектов ServiceAccount, в том числе в системных пространствах имён;
- Создание, изменение и удаление объектов Role и ClusterRole;
- Создание, изменение и удаление объектов ClusterRoleBinding;
- Использование команд attach и exec к Pod, а также добавление ephemeral-контейнеров (запросы и модификация объектов pods/exec, pods/attach, pods/ephemeralcontainers).

События, которые намеренно не фиксируются политикой аудита (из-за высокой частоты обновления соответствующих объектов):

- Операции с объектами типов Endpoints, EndpointSlice и Event;
- Операции с объектами Lease (выбор лидера raft в компонентах платформы);
- Действия с ConfigMap, используемыми для выбора лидера raft (cert-manager-cainjector-leader-election, cert-manager-controller, ingress-nginx и др.);
- Операции с объектами VerticalPodAutoscalerCheckpoints;
- Операции PATCH для объектов VerticalPodAutoscaler, выполняемые сервисным аккаунтом d8-vertical-pod-autoscaler-recommender;
- Операции с объектами UptimeHookProbes;
- Любые операции в пространстве имён d8-upmeter.

Эти политики активны по умолчанию. Чтобы отключить их, установите параметр basicAuditPolicyEnabled в значение false.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

Настройка собственной политики аудита

Чтобы создать расширенную политику аудита API Kubernetes, выполните следующие шаги:

1. Включите параметр auditPolicyEnabled в настройках модуля control-plane-manager:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    apiserver:
      auditPolicyEnabled: true
```

2. Создайте Secret kube-system/audit-policy, содержащий YAML-файл политики в формате Base64:

```

apiVersion: v1
kind: Secret
metadata:
  name: audit-policy
  namespace: kube-system
data:
  audit-policy.yaml: <Base64>

```

Пример содержимого файла `audit-policy.yaml` с минимальной рабочей конфигурацией:

```

apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  omitStages:
  - RequestReceived

```

Источники с подробной информацией по настройке возможного содержимого файла `audit-policy.yaml`:

- официальная документация Kubernetes;
- статья из блога компании «Флант» на Habr;
- код скрипта-генератора, используемого в GCE.

Как формировать содержимое файла `audit-policy.yaml`

Документация Kubernetes Структура полей ресурса Policy

Политика аудита в Kubernetes задается в файле формата YAML и состоит из набора правил, определяющих какие события и с каким уровнем детализации будут зафиксированы в журнале. Файл имеет следующую структуру:

```

apiVersion: audit.k8s.io/v1 # Версия API, используемая для политики аудита
kind: Policy                 # Тип ресурса – постоянно Policy
rules:                       # Набор правил для аудита
- level:                     # Уровень детализации логов. Обязательное поле
  users:                     # Список пользователей, чьи действия логируются
  userGroups:                # Группы пользователей (например, system:serviceaccounts)
  verbs:                     # Действия/операции, которые логируются (create, update,
delete и т.д.)
  resources:                 # Ресурсы Kubernetes для которых применяется правило
  namespaces:                # Пространства имен (namespaces), которые охватывает
                             правило

```

Массив `rules` описывает правила аудита. Каждое правило содержит следующие поля:

- `level` — уровень детализации логируемого события. Возможные значения (от наиболее детализированного к наименее):
 - `None` — не логировать вообще,
 - `Metadata` — только метаданные запроса (кто, когда, что, откуда; без содержимого объекта),
 - `Request` — также сохраняет тело запроса (только для запросов на изменение),
 - `RequestResponse` — сохраняет и тело запроса и содержимое ответа.
- `users` — перечень имен пользователей, на которых действует правило (например, `["admin"]`). Если это сервисные аккаунты, то имя обычно выглядит как `system:serviceaccount:<namespace>:<serviceaccount-name>`. Для обычных пользователей имя зависит от настроек системы аутентификации. Для объектов `deckhouse.io/v1` `Users` в качестве имени используется `email`.
- `userGroups` — группы пользователей (например, `["system:authenticated"]`). После аутентификации `kube-apiserver` присваивает каждому пользователю список групп (например, все аутентифицированные пользователи — часть `system:authenticated`, сервисные аккаунты — в дополнительных группах). Если запрос пришёл от пользователя, который состоит хотя бы в одной из групп, указанных в `userGroups`, то правило применяется к этому запросу.

Встроенные группы Kubernetes:

- `system:authenticated` — все, кто прошёл аутентификацию.
- `system:unauthenticated` — запросы от анонимных пользователей.
- `system:serviceaccounts` — все сервисные аккаунты всех namespaces.
- `system:serviceaccounts:<namespace>` — сервисные аккаунты в конкретном namespace.
- `verbs` — список операций API (`get`, `list`, `create`, `delete` и т.д.).
- `resources` — массив целевых ресурсов:
 - `group` — API-группа (например, `"apps"`, `"batch"`, `"` для `core`),
 - `resources` — виды ресурсов (например, `["pods", "deployments"]`). Полный перечень ресурсов и их групп может быть получен с помощью команды `kubectl api-resources`.
- `namespaces` — массив пространств имен (namespaces), в которых применяется правило.
- `nonResourceURLs` — набор URL-путей, подлежащих аудиту. Символ `*` разрешен, но только в качестве полного, последнего шага пути. Примеры:
 - `/metrics` — регистрировать запросы к метрикам `apiserver`,
 - `/healthz*` — регистрировать все `health`-запросы.

Примеры

Логирование всех запросов от всех аутентифицированных пользователей

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
```



Внимание! Не рекомендуется в production-средах. При большом потоке событий возможна повышенная нагрузка на дисковую подсистему control-plane узлов кластера.

Пример log-записи (фрагмент):

```
{
  "stage": "ResponseComplete",
  "requestURI": "/healthz",
  "verb": "get",
  "user": {
    "username": "kube-apiserver-kubelet-client",
    "groups": [
      "kubeadm:cluster-admins",
      "system:authenticated" // <- все, кто прошел аутентификацию
    ]
  },
  ...
}
```

Логирование всех запросов не от сервисных аккаунтов

```

apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
    users: []
    omitStages: []
    # Исключаем сервисные аккаунты через deny-правило
  - level: None
    userGroups: ["system:serviceaccounts"]

```

Пример log-записи (фрагмент):

```

{
  "user": {
    "username": "user@example.com",
    "groups": ["admins", "system:authenticated"]
  },
  ...
}

```

Логирование всех операций с типовыми ресурсами

```

apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    resources:
      - group: "apps"
        resources: ["deployments", "daemonsets", "statefulsets", "replicasets"]
      - group: "batch"
        resources: ["jobs", "cronjobs"]
      - group: ""
        resources: ["*"]

```

Работа с лог-файлом аудита

Предполагается, что на master-узлах кластера Deckhouse Kubernetes Platform установлен инструмент для отслеживания лог-файла `/var/log/kube-audit/audit.log`: `log-shipper`, `promtail` или `filebeat`.

Параметры ротации логов в файле предустановлены и не подлежат изменению:

- максимальный размер файла: 1000 МБ,
- максимальная глубина записи: 30 дней.

В зависимости от настроек политики и объема запросов к API-серверу количество записей может быть очень большим. В таких условиях глубина хранения может составлять менее 30 минут.



Наличие неподдерживаемых опций или опечаток в конфигурационном файле может привести к ошибкам при запуске API-сервера.

В случае возникновения проблем с запуском API-сервера выполните следующие шаги:

1. Вручную удалите параметры `--audit-log-*` из манифеста `/etc/kubernetes/manifests/kube-apiserver.yaml`;
2. Перезапустите API-сервер с помощью следующей команды:

```
crictrl stop $(crictrl pods --name=kube-apiserver -q)

# Альтернативный вариант (в зависимости от используемого CRI).
docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')
```

3. После перезапуска исправьте Secret или удалите его с помощью следующей команды:

```
d8 k -n kube-system delete secret audit-policy
```

Перенаправление лог-файла аудита в stdout

По умолчанию лог аудита сохраняется в файл `/var/log/kube-audit/audit.log` на master-узлах. При необходимости вы можете перенаправить его вывод в stdout процесса `kube-apiserver` вместо файла, установив параметр `apiserver.auditLog.output` модуля `control-plane-manager` в значение `Stdout`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

В этом случае лог будет доступен в stdout контейнера `kube-apiserver`.

Далее, используя встроенный механизм логирования в DKP, вы можете настроить сбор и отправку логов в собственную систему безопасности.

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Документация модуля `virtualization`

Рантайм-аудит

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства поиска угроз безопасности за счёт анализа событий ядра Linux и аудита событий Kubernetes API. DKP позволяет:

- находить угрозы в окружениях, анализируя приложения и контейнеры;
- обнаруживать попытки применения уязвимостей из базы CVE и признаки запуска криптовалютных майнеров;
- выявлять угрозы Kubernetes, включая:
 - оболочки командной строки, запущенные в контейнерах или подах;
 - контейнеры, работающие в привилегированном режиме;
 - монтирование небезопасных путей в контейнеры (например, `/proc`);
 - попытки чтения секретных данных (например, из `/etc/shadow`).

Источники данных для рантайм-аудита

DKP использует два основных источника событий:

- события ядра Linux — с помощью eBPF-драйвера для системы обнаружения угроз Falco;
- события аудита API Kubernetes — через интеграцию с механизмом Kubernetes auditing и вебхук-интерфейс.

Подробности об архитектуре рантайм-аудита можно найти в разделе «Архитектура рантайм-аудита».

Минимальные требования

Для получения событий ядра требуется:

- ядро Linux версии 5.8 или выше;
- поддержка eBPF. Проверить наличие поддержки можно одним из следующих способов:
 - убедитесь в наличии файла `/sys/kernel/btf/vmlinux`:

```
ls -lah /sys/kernel/btf/vmlinux
```

- убедитесь, что включён параметр `CONFIG_DEBUG_INFO_BTF`:

```
grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*
```

На каждом узле кластера запускаются агенты Falco, которые потребляют ресурсы в зависимости от количества применяемых правил или собираемых событий.



На некоторых системах могут не работать пробы eBPF.

Включение рантайм-аудита

Чтобы включить рантайм-аудит, выполните следующие шаги:

1. Убедитесь, что узлы соответствуют минимальным требованиям.
2. Включите модуль `runtime-audit-engine`, используя следующую конфигурацию:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: runtime-audit-engine
spec:
  enabled: true
```

3. **(Опционально)** Если control plane в кластере не управляется DKP при помощи `control-plane-manager`, настройте вебхук аудита API Kubernetes вручную.

Все доступные параметры рантайм-аудита доступны в разделе документации модуля `runtime-audit-engine`.

Настройка вебхука API Kubernetes вручную



Настройка вебхука не требуется, если включён модуль `control-plane-manager`. В этом случае при включении модуля `runtime-audit-engine` настройки сбора событий аудита API Kubernetes применяются автоматически.

Чтобы настроить вебхук на получение событий аудита от `kube-apiserver`, выполните следующие шаги:

1. Создайте файл `kubeconfig` для вебхука с адресом `https://127.0.0.1:9765/k8s-audit` и данными сертификата (`ca.crt`) из Secret `d8-runtime-audit-engine/runtime-audit-engine-webhook-tls`.

Пример:

```

apiVersion: v1
kind: Config
clusters:
  - name: webhook
    cluster:
      certificate-authority-data: BASE64_CA
      server: "https://127.0.0.1:9765/k8s-audit"
users:
  - name: webhook
contexts:
  - context:
      cluster: webhook
      user: webhook
      name: webhook
    current-context: webhook

```

2. Укажите путь к созданному файлу конфигурации с помощью флага `--audit-webhook-config-file` в манифесте `kube-apiserver`.
3. (**Опционально**) Чтобы собирать события аудита API Kubernetes не только из системных, но и пользовательских неймспейсов, настройте политики аудита.

Работа с правилами аудита

Для анализа событий рантайм-аудита используются правила, определяющие критерии подозрительного поведения. В DKP предусмотрены:

- **встроенные правила**, включая:
 - правила для аудита Kubernetes (располагаются в контейнере `falco` по пути `/etc/falco/k8s_audit_rules.yaml`);
 - правила, удовлетворяющие требованиям приказа ФСТЭК России №118 от 4 июля 2022 г. «Требования по безопасности информации к средствам контейнеризации» (`fstec`, в формате кастомного ресурса `FalcoAuditRules`);

Чтобы настроить список встроенных правил, используйте параметр `settings.builtInRulesList` модуля `runtime-audit-engine`.

- **пользовательские правила**, которые задаются через кастомный ресурс `FalcoAuditRules`.

Подробности о работе правил рантайм-аудита можно найти в разделе «Архитектура рантайм-аудита».

Добавление пользовательского правила

Чтобы добавить правило, создайте ресурс `FalcoAuditRules` с необходимыми условиями. Используйте синтаксис условий Falco. Агенты Falco автоматически применят созданное правило.

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: ownership-permissions
spec:
  rules:
    - macro:
        name: spawned_process
        condition: (evt.type in (execve, execveat) and evt.dir=<=)
    - rule:
        name: Detect Ownership Change
        desc: detect file permission/ownership change
        condition: >
          spawned_process and proc.name in (chmod, chown) and proc.args contains
          "/tmp/"
        output: >
          The file or directory below has had its permissions or ownership changed
          (user=%user.name
            command=%proc.cmdline file=%fd.name parent=%proc.pname
            pcmdline=%proc.pcmdline gparent=%proc.aname[2])
        priority: Warning
        tags: [filesystem]

```

Дополнительные примеры правил можно найти на следующих ресурсах:

- официальный репозиторий правил Falco;
- раздел с правилами Falco на Artifact Hub.

Применение стороннего правила

Поскольку структура правил Falco отличается от схемы кастомных ресурсов DKP, сторонние правила из интернета необходимо сконвертировать в ресурс FalcoAuditRules перед применением.

Используйте следующий скрипт для конвертации:

```

git clone github.com/deckhouse/deckhouse
cd deckhouse/ee/modules/runtime-audit-engine/hack/far-converter
go run main.go -input /path/to/falco/rule_example.yaml > ./my-rules-cr.yaml

```

Пример результата работы скрипта:

- Изначальное правило:

```

# /path/to/falco/rule_example.yaml
- macro: spawned_process
  condition: (evt.type in (execve, execveat) and evt.dir=<=)

- rule: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
  desc: "This rule detects an attempt to exploit a container escape
  vulnerability in the Linux Kernel."
  condition: container.id != "" and proc.name = "unshare" and spawned_process
  and evt.args contains "mount" and evt.args contains "-o rdma" and evt.args
  contains "/release_agent"
  output: "Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
  (user=%user.loginname uid=%user.loginuid command=%proc.cmdline
  args=%proc.args)"
  priority: CRITICAL
  tags: [process, mitre_privilege_escalation]

```

- Ресурс с правилом после конвертации:


```
# ./my-rules-cr.yaml
apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: rule-example
spec:
  rules:
    - macro:
        name: spawned_process
        condition: (evt.type in (execve, execveat) and evt.dir=<=)
    - rule:
        name: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
        condition: container.id != "" and proc.name = "unshare" and
spawned_process and evt.args contains "mount" and evt.args contains "-o rdma"
and evt.args contains "/release_agent"
        desc: This rule detects an attempt to exploit a container escape
vulnerability in the Linux Kernel.
        output: Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-
0492) (user=%user.loginname uid=%user.loginuid command=%proc.cmdline
args=%proc.args)
        priority: Critical
        tags:
          - process
          - mitre_privilege_escalation
```

Сбор логов и оповещения

DKP экспортирует события рантайм-аудита в формате метрик Prometheus, по которым можно настроить сбор логов и оповещения с помощью ресурсов модулей `log-shipper` и `observability`. Это позволяет:

- подключить внешнее хранилище для сбора логов (например, Loki или Elasticsearch);
- настроить оповещения о критических событиях.

Настройка сбора логов и событий

Все события рантайм-аудита выводятся в stdout. Для сбора и отправки событий в хранилище логов создайте ресурс `ClusterLoggingConfig`, следуя примеру:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: falco-events
spec:
  destinationRefs:
    - xxxx
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: "kubernetes.io/metadata.name"
            operator: In
            values: [d8-runtime-audit-engine]
  labelFilter:
    - operator: Regex
      values: ["\\{.*"} # Для сбора логов только в формате JSON.
      field: "message"
  type: KubernetesPods
```

Настройка оповещений о критических событиях

Для создания оповещений о критических событиях:

1. Убедитесь, что включён модуль `observability`.
2. Создайте ресурс `ClusterObservabilityMetricsRulesGroup` или `ObservabilityMetricsRulesGroup`:
 - `ClusterObservabilityMetricsRulesGroup` — предназначен для cluster-wide-правил, применяемых к системным неймспейсам;
 - `ObservabilityMetricsRulesGroup` — предназначен для правил, работающих в рамках конкретного неймспейса.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: falco-critical-alerts
spec:
  groups:
    - name: falco-critical-alerts
      rules:
        - alert: FalcoCriticalAlertsAreFiring
          for: 1m
          annotations:
            description: |
              There is a suspicious activity on a node {{ $labels.node }}.
              Check your events journal for more details.
            summary: Falco detected a critical security incident.
          expr: |
            sum by (node)
              (rate(falcosecurity_falcosidekick_falco_events_total{priority="Critical"}[5m]) > 0)
```

Просмотр метрик

Для получения Prometheus-метрик используйте PromQL-запрос `falcosecurity_falcosidekick_falco_events_total{}`:

```
d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
  curl -s "http://127.0.0.1:9090/api/v1/query?
  query=falcosecurity_falcosidekick_falco_events_total" | jq
```

Отладка и эмуляция событий

Для отладки и эмуляции событий рантайм-аудита в DKP можно использовать:

- утилиту `event-generator`;
- HTTP-эндпоинт `/test` сервиса `falcosidekick`.

Включение логов для отладки

В Falco по умолчанию используется отладочный уровень логирования `debug`.

В Falcosidekick по умолчанию отладочное логирование отключено. Для включения установите параметр `spec.settings.debugLogging` в `true`, следуя примеру:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: runtime-audit-engine
spec:
  version: 1
  enabled: true
  settings:
    debugLogging: true
```

Настройка эмуляции событий

Falco

Утилита `event-generator` позволяет генерировать различные подозрительные действия (например, системные вызовы или события аудита API Kubernetes).

Используйте следующую команду для запуска тестового набора событий в кластере Kubernetes:

```
d8 k run falco-event-generator --image=falcosecurity/event-generator run
```

Если вам нужно симитировать определённое действие, обратитесь к руководству утилиты.

Falcosidekick

Чтобы имитировать отправку тестовых событий для сервиса `falcosidekick`, используйте HTTP-эндпоинт `/test`:

1. Создайте тестовое событие, выполнив следующую команду:

```
nsenter -t $(pidof falcosidekick) curl -X POST -H "Content-Type: application/json" -H "Accept: application/json" http://localhost:2801/test
```

2. Проверьте метрику события:

```
d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
  curl -s "http://127.0.0.1:9090/api/v1/query?
  query=falcosecurity_falcosidekick_falco_events_total" \
  | jq '.data.result[] | select (.metric.priority_raw == "debug")'
```

Пример вывода:

```
{
  "metric": {
    "__name__": "falcosecurity_falcosidekick_falco_events_total",
    "container": "kube-rbac-proxy",
    "hostname": "falcosidekick",
    "instance": "192.168.208.7:4212",
    "job": "runtime-audit-engine",
    "node": "dev-master-0",
    "priority": "1",
    "priority_raw": "debug",
    "rule": "Test rule",
    "source": "internal",
    "tier": "cluster"
  },
  "value": [
    1744234729.799,
    "1"
  ]
}
```

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Документация модуля `log-shipper`
- Документация модуля `observability`
- Документация модуля `runtime-audit-engine`

Политики безопасности

Deckhouse Kubernetes Platform (DKP) позволяет управлять безопасностью приложений в кластере с помощью набора политик, соответствующих модели Kubernetes Pod Security Standards и дополнительно расширяемых через встроенные механизмы DKP.

Для реализации политик безопасности в DKP используется Gatekeeper.

Применение Pod Security Standards

В DKP поддерживаются три уровня политики безопасности:

- `privileged` — неограничивающая политика с максимально широким уровнем разрешений;
- `baseline` — минимально ограничивающая политика, которая предотвращает наиболее известные и популярные способы повышения привилегий. Позволяет использовать стандартную (минимально заданную) конфигурацию пода;
- `restricted` — политика со значительными ограничениями. Предъявляет самые жесткие требования к подам.

Политика по умолчанию

Используемая по умолчанию политика определяется следующим образом:

- в версиях DKP до v1.55 политика по умолчанию — `privileged`;
- начиная с версии DKP v1.55, политика по умолчанию — `baseline`.

i

При обновлении DKP на версию v1.55 или выше политика по умолчанию не изменится автоматически.

Назначение политики

Варианты назначения политики:

- глобально — с помощью параметра `settings.podSecurityStandards.defaultPolicy` модуля `admission-policy-engine`;
- для конкретного неймспейса — с помощью лейбла `security.deckhouse.io/pod-policy=<POLICY_NAME>`.

Пример команды для назначения политики `restricted` на все поды в неймспейсе `my-namespace`:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
```

Режимы применения политики

Допустимые режимы применения политик:

- `deny` — запрещает выполнений действий.
- `dryrun` — не влияет на выполнение действий и используется для отладки. Информацию о событиях можно посмотреть в Grafana или в консоли с помощью команды `d8 k` или `kubectl`.
- `warn` — работает как `dryrun`, но дополнительно выводит предупреждение с указанием причины, по которой бы произошёл запрет действия в режиме `deny`.

По умолчанию, политики Pod Security Standards в DKP применяются в режиме `deny`. В этом режиме поды приложений, не удовлетворяющие политикам, не могут быть запущены в кластере.

Как и в случае с назначением политик, режим их применения можно задать:

- глобально — с помощью параметра `settings.podSecurityStandards.enforcementAction` модуля `admission-policy-engine`;
- для конкретного неймспейса — с помощью лейбла `security.deckhouse.io/pod-policy-action=<POLICY_ACTION>`.

Пример команды для установки режима `warn` на все поды в неймспейсе `my-namespace`:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
```

Расширение политики

Вы можете расширить политики `baseline` и `restricted` с помощью шаблонов Gatekeeper, добавив необходимые проверки к уже существующим.

Чтобы расширить политику, выполните следующее:

1. Создайте шаблон проверки с помощью `ConstraintTemplate`.
2. Примените созданный шаблон к политике `baseline` или `restricted`.

Пример шаблона для проверки адреса репозитория с образом контейнера:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }
```

Пример применения шаблона к политике `restricted`:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"
```

В этом примере проверяется адрес репозитория в поле `image` у всех подов в неймспейсе с лейблом `security.deckhouse.io/pod-policy: restricted`. Если адрес в поле `image` создаваемого пода начинается не с `mycompany.registry.com`, под создан не будет.

Вспомогательные ресурсы при создании расширенных политик:

- примеры пользовательских политик;
- документация Gatekeeper с информацией о шаблонах и языке политик;
- библиотека Gatekeeper с примерами шаблонов проверок.

Операционные политики

DKP предоставляет механизм создания операционных политик с помощью `OperationPolicy`. В операционных политиках задаются требования к объектам в кластере: допустимые репозитории, требуемые ресурсы, наличие проб и т. д.

Команда разработки DKP рекомендует установить следующую политику с минимально необходимым набором требований:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.ru
    requiredResources:
      limits:
        - memory
      requests:
        - cpu
        - memory
    disallowedImageTags:
      - latest
    requiredProbes:
      - livenessProbe
      - readinessProbe
    maxRevisionHistoryLimit: 3
    imagePullPolicy: Always
    priorityClassNames:
      - production-high
      - production-low
    checkHostNetworkDNSPolicy: true
    checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/enabled: "true"
```

Эта политика задаёт базовые операционные требования к объектам в кластере, включая разрешённые container registries контейнеров, обязательные ресурсы и пробы, запрет на использование образов с тегом `latest`, допустимые классы приоритетов и другие настройки, повышающие безопасность и стабильность работы приложений.

Чтобы назначить данную операционную политику, примените лейбл `operation-policy.deckhouse.io/enabled=true` к необходимому неймспейсу:

```
d8 k label ns my-namespace operation-policy.deckhouse.io/enabled=true
```

Политики безопасности

Используя `SecurityPolicy`, вы можете создавать политики безопасности, задающие ограничения

на поведение контейнеров в кластере: доступ к host-сетям, привилегии, использование AppArmor и т. д.



Подробная информация о настройках безопасности подов и контейнеров (таких как `hostNetwork`, `hostPID`, `hostIPC` и других), доступные значения, а также практические советы доступны на странице «Параметры безопасности подов и контейнеров».

Пример политики безопасности:

```

---
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
      - driver: vmware
    allowedHostPorts:
      - max: 4000
        min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
      - unconfined
    allowedUnsafeSysctls:
      - kernel.*
    allowedVolumes:
      - hostPath
      - projected
    fsGroup:
      ranges:
        - max: 200
          min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
      - ALL
    runAsGroup:
      ranges:
        - max: 500
          min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
        - max: 200
          min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
        - my_profile.json
      allowedProfiles:
        - Localhost
    supplementalGroups:
      ranges:
        - max: 133
          min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          enforce: mypolicy

```

Чтобы назначить данную политику безопасности, примените лейбл `enforce: "mypolicy"` к

необходимому неймспейсу.

Частичное применение политик

Чтобы применить отдельные политики безопасности, не отключая весь предустановленный набор, выполните следующие шаги:

1. Добавьте на необходимый неймспейс лейбл `security.deckhouse.io/pod-policy: privileged`, чтобы отключить встроенный набор политик.
2. Создайте SecurityPolicy, соответствующий уровню `baseline` или `restricted`. В секции `policies` укажите только необходимые вам настройки безопасности.
3. Добавьте на неймспейс дополнительный лейбл, соответствующий селектору `namespaceSelector` в SecurityPolicy.

Пример конфигурации SecurityPolicy, соответствующий уровню `baseline`:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - AUDIT_WRITE
      - CHOWN
      - DAC_OVERRIDE
      - FOWNER
      - FSETID
      - KILL
      - MKNOD
      - NET_BIND_SERVICE
      - SETFCAP
      - SETGID
      - SETPCAP
      - SETUID
      - SYS_CHROOT
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    selinux:
      - type: ""
      - type: container_t
```

```

- type: container_init_t
- type: container_kvm_t
- type: container_engine_t
seccompProfiles:
  allowedProfiles:
    - RuntimeDefault
    - Localhost
    - undefined
    - ''
  allowedLocalhostFiles:
    - '*'
match:
  namespaceSelector:
  labelSelector:
  matchLabels:
    operation-policy.deckhouse.io/baseline-enabled: "true"

```

Пример конфигурации SecurityPolicy, соответствующий уровню `restricted`:

```

apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: restricted
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
      - min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
  requiredDropCapabilities:

```

```

- ALL
runAsUser:
  rule: MustRunAsNonRoot
seLinux:
  - type: ""
  - type: container_t
  - type: container_init_t
  - type: container_kvm_t
  - type: container_engine_t
seccompProfiles:
  allowedProfiles:
    - RuntimeDefault
    - Localhost
  allowedLocalhostFiles:
    - '*'
match:
  namespaceSelector:
  labelSelector:
  matchLabels:
    operation-policy.deckhouse.io/restricted-enabled: "true"

```

Кастомные ресурсы Gatekeeper

Gatekeeper предоставляет расширенные возможности для модификации ресурсов Kubernetes с помощью настраиваемых политик (mutation policies). Эти политики описываются через следующие кастомные ресурсы:

- AssignMetadata — для изменения секции `metadata` в ресурсе;
- Assign — для изменения других полей, кроме `metadata`;
- ModifySet — для добавления или удаления значений из списка, например, аргументов для запуска контейнера;
- AssignImage — для изменения параметра `image` ресурса.

Подробнее о возможности изменения ресурсов Kubernetes с помощью настраиваемых политик можно прочитать в документации Gatekeeper.

Примеры пользовательских политик Gatekeeper

Здесь приведены примеры политик Gatekeeper, с помощью которых вы можете расширить стандартные механизмы безопасности кластера.

Запрет на удаление узла без указанного лейбла

i Операции `DELETE` обрабатываются Gatekeeper по умолчанию.

Вы можете создать политику Gatekeeper, запрещающую удаление узла без специального лейбла.

В примере ниже используется поле `oldObject` для проверки лейблов удаляемого узла:

```

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Удаление Node запрещено. Добавьте лейбл %q=%q.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
      ---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"

```

Запрет на выполнение операций `exec` и `attach` в определённые поды

Вебхук модуля `admission-policy-engine` направляет запросы `CONNECT` для `Pods/exec` и `Pods/attach` через Gatekeeper. Это позволяет создавать пользовательские политики для разрешения или запрета операций `kubectl exec` и `kubectl attach`.

Встроенная политика для подов с лейблом `heritage: deckhouse`

Для защиты системных компонентов под управлением Deckhouse в модуле `admission-policy-engine` предусмотрена встроенная политика `D8DenyExecHeritage`, которая запрещает выполнение операций `kubectl exec` и `kubectl attach` во все поды с лейблом `heritage: deckhouse`.

Политика не распространяется на следующих пользователей, которым разрешены операции `kubectl exec` и `kubectl attach` в поды с лейблом `heritage: deckhouse`:

- `system:sudouser`;
- сервисные аккаунты из неймспейсов `d8-*` (`system:serviceaccount:d8-*`);
- сервисные аккаунты из неймспейсов `kube-*` (`system:serviceaccount:kube-*`).

Пример пользовательской политики

Вы можете создать собственную политику Gatekeeper для запрета операций `kubectl exec` и `kubectl attach` в определённых неймспейсах. В примере ниже используются `input.review.operation` и `input.review.resource.resource` для проверки операций `CONNECT`:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource предпочтительнее, но на всякий случай падаем в
        subResource
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource",
            input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
```

```

    }

    violation[{"msg": msg}] {
      is_connect
      is_exec_or_attach
      is_forbidden_namespace
      msg := sprintf("Exec/attach запрещён в неймспейсе %q",
[input.review.namespace])
    }
  ---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
  name: deny-exec-in-namespaces
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: ["*"]
        kinds: ["*"]
    scope: Namespaced
  parameters:
    forbiddenNamespaces:
      - production
      - staging

```

Ключевые данные и проверки, доступные при валидации операций `CONNECT`:

- Используйте `input.review.operation == "CONNECT"` для проверки операций `CONNECT`.
- Информация о пользователе доступна в `input.review.userInfo.username` и `input.review.userInfo.groups`.
- Неймспейс доступен в `input.review.namespace`.

Проверка подписи образов

DKP поддерживает проверку подписей образов контейнеров с помощью инструмента Cosign. Проверка подписи образов контейнеров позволяет убедиться в их целостности (что образ не был изменен после его создания) и подлинности (что образ был создан доверенным источником). Включить проверку подписи образов контейнеров в кластере можно с помощью параметра `policies.verifyImageSignatures SecurityPolicy`.

При подписании образа, в хранилище образов контейнеров создается специальный тег, который содержит подпись образа.

Подпись формируется для хеш-суммы образа (image digest). Например, для образа `my-hero/app:latest` с хешем `sha256:abc123EXAMPLE`, в хранилище образов появится образ `my-hero/app:sha256-abc123EXAMPLE.sig`. Таким образом, процесс подписи образа заключается в расчете и публикации дополнительного тега в хранилище образов контейнеров, без изменения существующего образа.

После подписания образа повторная его загрузка (push) в хранилище не требуется. Необходимо только авторизоваться в хранилище образов контейнеров с правами на запись.

Чтобы подписать образ с помощью Cosign, выполните следующее:

1. Убедитесь что используется Cosign версии 2 или ниже.

Проверить версию: `cosign version`.

```
cosign version
```

2. Сгенерируйте пару ключей (публичный и приватный):

```
cosign generate-key-pair
```

3. Подпишите образ в хранилище образов контейнеров с помощью сгенерированного приватного ключа:

```
cosign sign --key <KEY> <REGISTRY_IMAGE_PATH>
```

Здесь:

- `<REGISTRY_IMAGE_PATH>` — путь к образу, который нужно указать при запуске, например: `registry.private.ru/labs/application/image:latest`.

Чтобы включить проверку подписи образов контейнеров в кластере DKP:

1. Используйте параметр `policies.verifyImageSignatures` `SecurityPolicy`, указав сгенерированный публичный ключ.

Пример конфигурации `SecurityPolicy` для проверки подписи образов контейнеров в хранилище `registry.private.ru`, размещенные по пути `/labs/application/`:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: verify-image-test
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          example-security-policy/enabled: true
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowRbacWildcards: true
    verifyImageSignatures:
      - publicKeys:
          - |-
            -----BEGIN PUBLIC KEY-----
            ...
            -----END PUBLIC KEY-----
        reference: registry.private.ru/labs/application/*
```

Название лейбла указанного в `match.namespaceSelector.labelSelector.matchLabels` может быть любым. Требуется лишь совпадение лейбла в селекторе политик и соответствующего неймспейса.

Более подробную информацию о использовании селекторов вы можете прочитать в описании настройки селекторов.

2. Создайте `OperationPolicy`, ограничивающий запуск подов со сторонних registry:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: test-operation-policy
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          example-operation-policy/enabled: "true"
  policies:
    allowedRepos:
      - registry.private.ru
```

Название лейбла указанного в `match.namespaceSelector.labelSelector.matchLabels` может быть любым. Требуется лишь совпадение лейбла в селекторе политик и соответствующего неймспейса.

Более подробную информацию о использовании селекторов вы можете прочитать в описании настройки селекторов.

3. Добавьте метку на неймспейс, где необходимо включить проверку подписи командой (укажите нужный неймспейс):

```
d8 k label ns <NAMESPACE> example-security-policy/enabled=true
```

4. Добавьте метку на неймспейс, где необходимо ограничить запуск подов со сторонних registry (укажите нужный неймспейс):

```
d8 k label ns <NAMESPACE> example-operation-policy/enabled=true
```

5. Для проверки работы механизма подписи образов разверните поды в неймспейсе, с подписанным и неподписанным образами (укажите нужный неймспейс):

```
d8 k -n <NAMESPACE> run signed-pod --image=<ПОДПИСАННЫЙ_ОБРАЗ>
d8 k -n <NAMESPACE> run unsigned-pod --image=<НЕПОДПИСАННЫЙ_ОБРАЗ>
```

Согласно данной политике, если адрес какого-либо образа контейнера совпадает со значением параметра `reference` и образ не подписан или подпись не соответствует указанным ключам, создание пода будет запрещено.

Пример вывода ошибки при создании пода с образом контейнера, не прошедшим проверку подписи:

```
[verify-image-signatures] Image signature verification failed: nginx:1.17.2
```

Использование альтернатив для управления политиками безопасности

Если вместо встроенного механизма управления политиками безопасности в кластере DKP используется альтернативное решение (например, Kyverno), настройте исключения для следующих неймспейсов:

- `kube-system`;
- все неймспейсы с префиксом `d8-*` (например, `d8-system`).

Без этих исключений политики могут блокировать или нарушать работу системных компонентов.

Дополнительные ресурсы

- Документация модуля `admission-policy-engine`

Сканирование контейнерных образов на уязвимости

Deckhouse Kubernetes Platform (DKP) предоставляет встроенное средство для автоматического поиска уязвимостей в контейнерных образах на основе проекта Trivy.

Поиск уязвимостей

DKP сканирует все контейнерные образы, используемые в подах кластера. Проверка охватывает:

- известные уязвимости (CVE) в используемых образах;
- соответствие CIS-стандартам (compliance-проверки).

Для сканирования используются как публичные базы уязвимостей, так и обогащённые данные из Astra Linux, ALT Linux и РЕД ОС.

Сканирование в пространствах имён

По умолчанию сканируется только пространство имён `default`.

Чтобы выполнить сканирование в конкретном пространстве имён, добавьте для него лейбл `security-scanning.deckhouse.io/enabled=""`.

Как только в кластере обнаруживается хотя бы одно пространство имён с указанным лейблом, сканирование пространства имён `default` прекращается. Чтобы снова включить сканирование для пространства имён `default`, добавьте для него лейбл следующей командой:

```
d8 k label namespace default security-scanning.deckhouse.io/enabled=""
```

В текущей версии функциональности ограничения перечня ресурсов для сканирования в пространстве имён не предусмотрено.

DKP сканирует **все ресурсы**, находящиеся в пространстве имён, помеченном лейблом `security-scanning.deckhouse.io/enabled=""`.

Условия запуска и алгоритм сканирования

Сканирование запускается:

- автоматически каждые 24 часа,
- при запуске компонентов с новыми образами контейнеров в пространствах имен, для которых включено сканирование (в частности, при появлении новых объектов).

Сканирование происходит согласно следующему алгоритму:

1. В пространстве имён с каждым просканированным ресурсом создаётся объект `VulnerabilityReport`.
2. Этот объект содержит аннотацию `trivy-operator.aquasecurity.github.io/report-ttl`, которая определяет срок жизни отчёта (по умолчанию - 24h).
3. По истечении этого срока объект `VulnerabilityReport` удаляется и сканирование запускается повторно.

Принудительный повтор сканирования

Чтобы запустить повторное сканирование ресурса вручную, воспользуйтесь одним из двух способов:

- Перезапишите аннотацию `trivy-operator.aquasecurity.github.io/report-ttl`, указав короткий срок жизни отчёта.

Пример команды:

```
d8 k annotate VulnerabilityReport -n <NAMESPACE> <REPORT_NAME> trivy-operator.aquasecurity.github.io/report-ttl=1s --overwrite
```

- Удалите объект `VulnerabilityReport` из пространства имён, где находится просканированный ресурс.

Пример команды:

```
d8 k delete VulnerabilityReport -n <NAMESPACE> <REPORT_NAME>
```

Для получения информации о том, как просматривать результаты сканирования, см. раздел «Сканирование контейнерных образов на уязвимости».

Управление сертификатами

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства управления TLS-сертификатами в кластере и поддерживает:

- заказ сертификатов во всех поддерживаемых источниках, таких как Let's Encrypt, HashiCorp Vault, Venafi;
- выпуск самоподписанных сертификатов;
- автоматический перевыпуск и контроль срока действия сертификатов;
- установку `cm-acme-http-solver` на master-узлы и выделенные узлы.

На этой странице описаны доступные в DKP возможности управления сертификатами, а также порядок работы с издателями сертификатов.

- i** Описание примеров настройки сертификатов, порядок работы с аннотацией `tls-acme` и способы безопасного использования учётных данных приведены на странице «Использование TLS-сертификатов».

Мониторинг

DKP экспортирует метрики в Prometheus, что позволяет отслеживать:

- срок действия сертификатов;
- статус перевыпуска сертификатов.

Роли доступа

В DKP предопределены несколько ролей для доступа к ресурсам:

Роль	Права доступа
User	Просмотр ресурсов Certificate и Issuer в доступных пространствах имён, а также глобальных ресурсов ClusterIssuer.
Editor	Управление ресурсами Certificate и Issuer в доступных пространствах имён.
ClusterEditor	Управление ресурсами Certificate и Issuer во всех пространствах имён.
SuperAdmin	Управление внутренними служебными объектами.

Работа с издателями сертификатов

В DKP по умолчанию поддерживаются следующие издатели сертификатов (ClusterIssuer):

- `letsencrypt` — выпускает TLS-сертификаты, используя публичный удостоверяющий центр Let's Encrypt и HTTP-валидацию по протоколу ACME. Используется для автоматического получения доверенных сертификатов, подходящих для большинства публичных сервисов. Подробное описание настроек доступно в официальной документации `cert-manager`.
- `letsencrypt-staging` — аналогичен `letsencrypt`, но использует тестовый сервер Let's Encrypt. Подходит для отладки конфигурации и проверки процесса выпуска сертификатов. Подробнее про тестовую среду Let's Encrypt можно прочитать в официальной документации.
- `selfsigned` — выпускает самоподписанные сертификаты. Используется в ситуациях, когда не требуется внешнее доверие к сертификату (например, для внутренних сервисов).
- `selfsigned-no-trust` — также выпускает самоподписанные сертификаты, но без автоматического добавления корневого сертификата в доверенные. Используется для ручного управления доверием.

В некоторых случаях вам могут понадобиться дополнительные виды ClusterIssuer:

- если вы хотите использовать сертификат от Let's Encrypt, но с DNS-валидацией через стороннего DNS-провайдера;
- когда необходимо использовать удостоверяющий центр (CA), отличный от Let's Encrypt. Все виды поддерживаемых удостоверяющих центров перечислены в документации `cert-manager`.

Добавление ClusterIssuer с валидацией DNS-01 через вебхук

Для подтверждения владения доменом через Let's Encrypt с помощью метода `DNS-01` необходимо, чтобы модуль `cert-manager` мог создавать TXT-записи в зоне DNS, связанной с доменом.

У модуля `cert-manager` есть встроенная поддержка популярных DNS-провайдеров, таких как AWS Route53, Google Cloud DNS, Cloudflare и других. Полный перечень доступен в официальной документации `cert-manager`.

Если провайдер не поддерживается напрямую, можно настроить вебхук и разместить в кластере собственный обработчик ACME-запросов, который будет выполнять нужные операции

для обновления DNS-записей.

Данный пример основан на использовании сервиса Yandex Cloud DNS:

1. Для обработки вебхука разместите в кластере сервис `Yandex Cloud DNS ACME webhook` согласно официальной документации.
2. Создайте ресурс `ClusterIssuer`, следуя примеру:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: yc-clusterissuer
  namespace: default
spec:
  acme:
    # Заменить этот адрес электронной почты на свой собственный.
    # Let's Encrypt будет использовать его, чтобы связаться с вами по поводу
истекающих
    # сертификатов и вопросов, связанных с вашей учетной записью.
    email: your@email.com
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      # Ресурс секретов, который будет использоваться для хранения закрытого
ключа аккаунта.
      name: secret-ref
    solvers:
      - dns01:
          webhook:
            config:
              # Идентификатор папки, в которой расположена DNS-зона.
              folder: <your-folder-ID>
              # Секрет, используемый для доступа к учетной записи сервиса.
              serviceAccountSecretRef:
                name: cert-manager-secret
                key: iamkey.json
              groupName: acme.cloud.yandex.com
              solverName: yandex-cloud-dns
```

Добавление ClusterIssuer, использующего собственный удостоверяющий центр (CA)

1. Сгенерируйте сертификат:

```
openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
```

2. В пространстве имён `d8-cert-manager` создайте секрет с произвольным именем, содержащий данные файлов сертификатов.

- Пример создания секрета с помощью команды `d8 k`:

```
d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --cert="rootCACert.pem"
```

- Пример создания секрета из YAML-файла (содержимое файлов сертификатов должно быть закодировано в Base64):

```

apiVersion: v1
data:
  tls.crt: <результат команды `cat rootCACert.pem | base64 -w0`>
  tls.key: <результат команды `cat rootCAKey.pem | base64 -w0`>
kind: Secret
metadata:
  name: internal-ca-key-pair
  namespace: d8-cert-manager
type: Opaque

```

3. Создайте ClusterIssuer с произвольным именем из созданного секрета:

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: inter-ca
spec:
  ca:
    secretName: internal-ca-key-pair    # Имя созданного секрета.

```

Теперь можно использовать созданный ClusterIssuer для получения сертификатов для всех компонентов DKP или конкретного компонента.

Например, чтобы использовать ClusterIssuer для получения сертификатов для всех компонентов DKP, укажите его имя в глобальном параметре `clusterIssuerName`:

```

spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
          mode: CertManager
          publicDomainTemplate: '%s.<public_domain_template>'
version: 1

```

Добавление Issuer и ClusterIssuer, использующих HashiCorp Vault для заказа сертификатов

Для настройки заказа сертификатов с помощью Vault используйте документацию HashiCorp.

После настройки PKI и включения авторизации в Kubernetes, выполните следующее:

1. Создайте ServiceAccount и скопируйте ссылку на его Secret:

```

d8 k create serviceaccount issuer

ISSUER_SECRET_REF=$(d8 k get serviceaccount issuer -o json | jq -r ".secrets[0].name")

```

2. Создайте ресурс Issuer:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    server: http://vault.default.svc.cluster.local:8200
    # Указывается на этапе конфигурации PKI.
    path: pki/sign/example-dot-com
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: $ISSUER_SECRET_REF
          key: token
EOF
```

3. Создайте ресурс Certificate для получения TLS-сертификата, подписанного CA Vault:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # Домены указываются на этапе конфигурации PKI в Vault.
  commonName: www.example.com
  dnsNames:
    - www.example.com
EOF
```

Дополнительные ресурсы

- Глобальные параметры
- Документация модуля `cert-manager`

Интеграция с ПО и решениями для безопасности

Deckhouse Kubernetes Platform (DKP) поддерживает интеграцию с Kaspersky Unified Monitoring and Analysis Platform (KUMA), единой системой мониторинга и анализа от «Лаборатории Касперского». В рамках интеграции события безопасности и журналы аудита из кластера передаются в KUMA для дальнейшего анализа.

Отправка логов в KUMA

Для отправки логов в систему KUMA настройте сбор и доставку логов на стороне DKP, используя следующие ресурсы:

- `ClusterLogDestination` — задаёт параметры хранилища логов;
- `ClusterLoggingConfig` — задаёт параметры сбора логов из кластера.

i На стороне KUMA настройте соответствующие ресурсы для приёма событий.

Примеры конфигурации `ClusterLogDestination` и `ClusterLoggingConfig`

- Отправка логов в формате JSON по UDP:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-udp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Замените при настройке.
    mode: UDP
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-udp-json

```

- Отправка логов в формате JSON по TCP:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Замените при настройке.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-tcp-json

```

- Отправка логов в формате CEF по TCP:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-cef
spec:
  type: Socket
  socket:
    extraLabels:
      cef.name: d8
      cef.severity: "1"
    address: IP_ADDRESS:PORT # Замените при настройке.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "CEF"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-cef

```

- Отправка логов в формате Syslog по TCP:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-syslog
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Замените при настройке.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "Syslog"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-syslog

```

- Отправка логов в Apache Kafka:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-kafka
spec:
  type: Kafka
  kafka:
    bootstrapServers:
      - kafka-address:9092 # Замените при настройке.
    topic: k8s-logs
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  destinationRefs:
    - kuma-kafka
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values:
        - kubelet.*
  type: File

```


Особенности настройки KCS

Kaspersky Container Security (KCS) для мониторинга среды выполнения требует расширенных прав доступа к ядру ОС. В частности, компонент `kcs-ih` (Image Hub) должен работать в привилегированном режиме. При этом некоторые механизмы управления привилегиями процессов на уровне ядра ОС Linux должны быть отключены. Подробнее — в документации Kaspersky Container Security.

Чтобы мониторинг среды выполнения с KCS работал в кластере DKP, назначьте лейбл `security.deckhouse.io/pod-policy=privileged` пространству имён, в котором размещаются компоненты KCS.

Пример команды для назначения лейбла:

```
d8 k label namespace kcs security.deckhouse.io/pod-policy=privileged
```

Исключения при антивирусном сканировании узлов

Если на узлах кластера DKP используются антивирусные средства (например, Kaspersky Endpoint Security (KESL)), вам может понадобиться исключить из анализа служебные директории Deckhouse, чтобы избежать ложных срабатываний.

Перечень служебных директорий Deckhouse (также доступен в CSV):

Директория	Назначение
<code>/etc/cni/</code> (любой узел)	Файлы настройки модуля CNI
<code>/etc/kubernetes</code> (любой узел)	Манифесты статических подов, файлы сертификатов PKI
<code>/mnt/kubernetes-data</code> (master-узел)	Существует только в кластерах, развёрнутых в облаке, когда используется отдельный диск для базы данных etcd
<code>/mnt/vector-data</code> (любой узел, модуль <code>log-shipper</code>)	Служебные данные статуса отправленных журналов
<code>/opt/cni/bin/</code> (любой узел)	Исполняемые файлы модуля CNI
<code>/opt/deckhouse/bin/</code> (любой узел)	Исполняемые файлы, необходимые для работы Deckhouse
<code>/var/lib/bashible</code> (любой узел)	Файлы настройки узла
<code>/var/lib/containerd</code> (любой узел)	Используется для хранения данных, связанных с работой CRI (например, containerd). Содержит слои образов контейнеров, снимки файловых систем контейнеров, метаданные контейнеров, логи и другую информацию контейнеров
<code>/var/lib/deckhouse/</code> (master-узел)	Файлы модулей Deckhouse, динамически загружаемых из хранилища образов
<code>/var/lib/etcd</code> (master-узел)	База данных etcd
<code>/var/lib/kubelet/</code> (любой узел)	Файлы настройки kubelet
<code>/var/lib/upmeter</code> (master-узел, модуль <code>upmeter</code>)	База данных модуля <code>upmeter</code>
<code>/var/log/containers</code> (любой узел)	Журналы контейнеров (при использовании containerd)
<code>/var/log/pods/</code> (любой узел)	Журналы всех контейнеров подов, запущенных на узле

Рекомендации по настройке KESL

Для корректной работы DKP при наличии установленного решения KESL выполните следующие шаги:

1. Отключите следующие задачи на стороне KESL:

- Firewall Management (ID: 12);
- Web Threat Protection (ID: 14);
- Network Threat Protection (ID: 17);
- Web Control (ID: 26).



В будущих версиях KESL список задач может отличаться.

2. Убедитесь, что ресурсы узлов соответствуют требованиям:

- DKP;
- KESL.

3. Для оптимизации производительности следуйте официальным рекомендациям «Лаборатории Касперского».

Дополнительные ресурсы

- Кастомные ресурсы модуля `log-shipper`
- Документация модуля `upmeter`

Автоматическое распространение секретов по пространствам имён

Deckhouse Kubernetes Platform (DKP) позволяет автоматически распространять секреты во все пространства имён кластера. Это помогает исключить ручное копирование секретов в рамках CI/CD-процессов, например, для скачивания образов или при работе с хранилищем Ceph RBD.

Как это работает

- Deckhouse отслеживает секреты в пространстве имён `default`, которым назначен специальный лейбл `secret-copier.deckhouse.io/enabled: ""`.
- Такие секреты автоматически копируются во все пространства имён кластера.
- При изменении оригинального секрета в пространстве `default` обновлённое содержимое также распространяется по всем пространствам имён кластера.
- При удалении оригинального секрета в пространстве `default` удаляются все его копии во всех пространствах имён.
- Если секрет был вручную изменён в одном из пространств имён, он будет автоматически перезаписан содержимым оригинального секрета из пространства `default`.
- При создании новых пространств имён в них автоматически копируются все секреты из `default` с лейблом `secret-copier.deckhouse.io/enabled: ""`.
- Каждую ночь выполняется повторная синхронизация всех секретов.

Несовместимость с режимом мультитенантности

Механизм автоматического распространения секретов несовместим с режимом мультитенантности.

В режиме мультитенантности создаются изолированные окружения для пользователей в проектах и автоматическое копирование секретов по всем пространствам имён может нарушить эту модель безопасности. Если в закрытом окружении пользователя окажутся чувствительные данные, это может привести к их утечке.

Если вам необходимо предоставить общий сертификат (например, WC-сертификат для внутреннего окружения) или общий токен доступа к registry, не используйте механизм автоматического распространения секретов. Вместо этого поместите такие секреты в шаблон проекта — администратор кластера должен определить их в конфигурации проекта.

Настройка автоматического распространения секретов

1. Создайте ресурс Secret в пространстве имён `default`.
2. Добавьте к нему лейбл `secret-copier.deckhouse.io/enabled: ""`.
3. Примените ресурс командой `d8 k apply`, после чего он будет автоматически распространён по всем пространствам имён.



Автоматически распространяются только секреты из пространства имён `default`. Секреты с лейблом `secret-copier.deckhouse.io/enabled: ""`, созданные в других пространствах имён, будут автоматически удалены.

Распространение секретов по выбранным пространствам имён

Чтобы копировать секрет не во все, а только в отдельные пространства имён, используйте аннотацию с label-селектором `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. В этом случае секрет будет скопирован только в пространства имён, соответствующие заданному селектору.

Мониторинг в Deckhouse Kubernetes Platform

Deckhouse Kubernetes Platform (DKP) предоставляет решение для мониторинга Kubernetes на базе **Prometheus** и **Grafana**. В кластере автоматически настраивается сбор метрик с узлов, подов и ключевых компонентов кластера (etcd, kube-apiserver, CoreDNS), после чего становятся доступны предустановленные дашборды для анализа использования CPU, памяти, дисков и сети.

Мониторинг кластера по умолчанию включён в наборах модулей `Default` и `Managed`.

Все компоненты, включая Prometheus и Alertmanager, работают в отказоустойчивом режиме и могут использоваться как в облачных средах, так и на bare-metal-серверах.

Принцип работы Prometheus рассмотрен в разделе Настройка системы сбора и хранения метрик.

В DKP реализовано несколько видов мониторинга:

- мониторинг аппаратных ресурсов;
- мониторинг Kubernetes;
- мониторинг Ingress;
- мониторинг control plane;
- мониторинг сетевого взаимодействия;
- расширенный мониторинг;
- мониторинг SLA кластера.

В DKP предусмотрена система оповещений (алертинг), поддерживающая отправку уведомлений о событиях, в том числе во внешние системы.

Мониторинг аппаратных ресурсов

Предусмотрено отслеживание загрузки аппаратных ресурсов кластера с графиками по утилизации:

- процессора;
- памяти;
- диска;
- сети.

Графики доступны с агрегацией:

- по подам;
- контроллерам;
- пространствам имен;
- узлам.

Мониторинг Kubernetes

Модуль `monitoring-kubernetes` предназначен для базового мониторинга узлов кластера.

Модуль обеспечивает безопасный сбор метрик и предоставляет базовый набор правил для мониторинга:

- текущей версии container runtime (docker, containerd) на узле и ее соответствия версиям, разрешенным для использования;
- общей работоспособности подсистемы мониторинга кластера (Dead man's switch);
- доступных файловых дескрипторов, сокетов, свободного места и inode;
- работы kube-state-metrics, node-exporter, kube-dns;
- состояния узлов кластера (NotReady, drain, cordon);
- состояния синхронизации времени на узлах;
- случаев продолжительного превышения CPU steal;
- состояния таблицы Conntrack на узлах;
- подов с некорректным состоянием (как возможное следствие проблем с kubelet) и др.

Мониторинг Ingress

Реализован сбор статистики для `ingress-nginx` в Prometheus с детальными метриками (время ответа, коды, география и др.), доступной в разных разрезах (namespace, vhost, ingress). Данные

визуализированы в Grafana с интерактивными дашбордами. Подробное описание доступно в разделе о мониторинге Ingress.

Модуль по умолчанию включён в наборах модулей `Default` и `Managed`.

Отключение сбора детализированной статистики Ingress-ресурсов

По умолчанию DKP собирает подробную статистику со всех Ingress-ресурсов в кластере. Этот процесс может приводить к высокой нагрузке системы мониторинга.

Для отключения сбора статистики добавьте лейбл `ingress.deckhouse.io/discard-metrics: "true"` к соответствующему пространству имён или Ingress-ресурсу.

- Пример отключения сбора статистики (метрик) для всех Ingress-ресурсов в пространстве имен `review-1`:

```
d8 k label ns review-1 ingress.deckhouse.io/discard-metrics=true
```

- Пример отключения сбора статистики (метрик) для всех Ingress-ресурсов `test-site` в пространстве имен `development`:

```
d8 k label ingress test-site -n development ingress.deckhouse.io/discard-metrics=true
```

Мониторинг control plane

Мониторинг control plane осуществляется с помощью модуля `monitoring-kubernetes-control-plane`, который организует безопасный сбор метрик и предоставляет базовый набор правил мониторинга следующих компонентов кластера:

- kube-apiserver;
- kube-controller-manager;
- kube-scheduler;
- kube-etcd.

Мониторинг кластера

DKP безопасно собирает метрики мониторинга и настраивает правила.

Возможности мониторинга DKP:

- мониторинг текущей версии container runtime (containerd) на узле и ее соответствия версиям, разрешенным для использования в DKP;
- мониторинг работоспособности подсистемы мониторинга кластера («Dead man's switch»);
- мониторинг доступных файловых дескрипторов, сокетов, свободного места и inode;
- мониторинг состояния узлов кластера (NotReady, drain, cordon);
- работы kube-state-metrics, node-exporter, kube-dns;
- мониторинг состояния синхронизации времени на узлах;
- мониторинг случаев продолжительного превышения CPU steal;
- мониторинг состояния таблицы Conntrack на узлах;
- мониторинг подов с некорректным состоянием (как возможное следствие проблем с kubelet);
- мониторинг компонентов control plane (реализуется модулем `monitoring-kubernetes-control-plane`).

Режим расширенного мониторинга

В DKP поддерживается режим расширенного мониторинга с помощью модуля `extended-monitoring`, который позволяет настроить:

- мониторинг секретов в Кластере (объекты Secret) и срока действия TLS-сертификатов в них;
- сбор событий в кластере Kubernetes в виде метрик;
- мониторинг доступности образов контейнеров в registry, используемых в контроллерах (Deployments, StatefulSets, DaemonSets, CronJobs);
- мониторинг объектов в пространствах имен с установленным лейблом `extended-monitoring.deckhouse.io/enabled=""`.

Модуль предусматривает отправку алертов по следующим метрикам:

- свободное место и inode на дисках узлов;
- утилизация узлов;

- доступность подов и образов контейнеров;
- истечение срока действия сертификатов;
- иные события кластера.

Алерты

Мониторинг в составе DKP включает уведомления о событиях. Стандартная поставка включает набор базовых предупреждений, охватывающих состояние кластера и его компоненты. Также остается возможность добавлять кастомные алерты.

Список всех доступных алертов системы мониторинга DKP приведён на отдельной странице документации.

Отправка алертов во внешние системы

DKP поддерживает отправку алертов с помощью Alertmanager:

- по протоколу SMTP;
- в PagerDuty;
- в Slack;
- в Telegram;
- посредством Webhook;
- по любым другим каналам, поддерживаемым в Alertmanager.

Примеры интеграции мониторинга DKP с внешними системами доступны в разделе Настройка интеграций.

Мониторинг SLA кластера

Оценка доступности в DKP осуществляется модулем `upmeter`.

Состав модуля `upmeter`:

- **agent** — работает на master-узлах и делает пробы доступности, отправляет результаты на сервер.
- **upmeter** — собирает результаты и поддерживает API-сервер для их извлечения.
- **front:**
 - **status** — показывает уровень доступности за последние 10 минут (требуется авторизации, но ее можно отключить).
 - **webui** — показывает дашборд со статистикой по пробам и группам доступности (требуется авторизации).
- **smoke-mini** — поддерживает постоянное *smoke-тестирование* с помощью StatefulSet.

Модуль отправляет около 100 показаний метрик каждые 5 минут. Это значение зависит от количества включенных модулей Deckhouse Kubernetes Platform.

Дополнительные ресурсы

- Список алертов мониторинга
- Настройка интеграции с внешними системами мониторинга
- Веб-интерфейс системы мониторинга
- Веб-интерфейс платформы
- Документация модуля `prometheus`

Настройка мониторинга сетевого взаимодействия и узлов кластера

Мониторинг сетевого взаимодействия

DKP может выполнять мониторинг сетевого взаимодействия между всеми узлами кластера, а также между узлами кластера и внешними хостами. При настроенном мониторинге, каждый узел два раза в секунду отправляет ICMP-пакеты на все другие узлы кластера (и на опциональные внешние узлы) и экспортирует данные в систему мониторинга.

Особенности мониторинга сетевого взаимодействия:

- автоматически проверяет доступность всех узлов кластера (и, опционально, внешних систем) с помощью ICMP (ping) — тестирование запускается каждые две секунды;
- все результаты экспортируются в формате метрик в систему мониторинга Prometheus;
- в комплекте — готовый дашборд для Grafana, где в реальном времени визуализируются текущая доступность, графики задержек и потенциальные проблемы с сетевой связностью

(подробнее о них читайте в разделе Grafana);

- позволяет быстро выявлять узлы с деградировавшей связностью и ускоряет реакцию на инциденты.

Модуль `monitoring-ping` отслеживает любые изменения поля `.status.addresses` узла. Если они обнаружены, срабатывает хук, который собирает полный список имен узлов и их адресов, и передает в `DaemonSet`, который заново создает поды. Таким образом, `ping` проверяет всегда актуальный список узлов.



Модуль `monitoring-ping` должен быть включен.

Добавление дополнительных IP-адресов для мониторинга

Для добавления дополнительных IP-адресов мониторинга используйте параметр `externalTargets` модуля `monitoring-ping`.

Пример конфигурации модуля:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings:
    externalTargets:
      - name: google-primary
        host: 8.8.8.8
      - name: yaru
        host: ya.ru
      - host: youtube.com
```



Поле `name` используется в Grafana для отображения связанных данных. Если поле `name` не указано, используется обязательное поле `host`.

Мониторинг узлов кластера

Чтобы включить мониторинг узлов кластера, необходимо включить модуль `monitoring-kubernetes`, если он не включен. Включить мониторинг кластера можно в веб-интерфейсе Deckhouse, или с помощью следующей команды:

```
d8 system module enable monitoring-kubernetes
```

Аналогичным способом можно включить модули `monitoring-kubernetes-control-plane` и `extended-monitoring`.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `extended-monitoring`
- Документация модуля `monitoring-kubernetes`
- Документация модуля `monitoring-kubernetes-control-plane`
- Документация модуля `monitoring-ping`

Запись данных Prometheus в долговременное хранилище

Prometheus поддерживает механизм `remote_write` для отправки данных из локального экземпляра Prometheus в отдельное долговременное хранилище (например, VictoriaMetrics). В DKP поддержка этого механизма реализована с помощью кастомного ресурса `PrometheusRemoteWrite`.

i Для VictoriaMetrics подробную информацию о способах передачи данные в vmagent можно получить в документации VictoriaMetrics.

Пример минимального PrometheusRemoteWrite

Ниже приведён пример минимальной конфигурации кастомного ресурса PrometheusRemoteWrite:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
```

Пример расширенного PrometheusRemoteWrite

Ниже приведён пример расширенной конфигурации кастомного ресурса PrometheusRemoteWrite:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
  basicAuth:
    username: username
    password: password
  writeRelabelConfigs:
    - sourceLabels: [__name__]
      action: keep
      regex: prometheus_build_.*|my_cool_app_metrics_.*
    - sourceLabels: [__name__]
      action: drop
      regex: my_cool_app_metrics_with_sensitive_data
```

С полным описанием полей кастомного ресурса можно ознакомиться в документации модуля `prometheus`.

Дополнительные ресурсы

- Кастомные ресурсы модуля `prometheus`

Настройка внешнего доступа

Подключение Prometheus к сторонней Grafana

У каждого `ingress-nginx-controller` есть сертификаты, при указании которых в качестве клиентских будет разрешено подключение к Prometheus. Все, что нужно, — создать дополнительный `Ingress`-ресурс.

i В приведенном ниже примере предполагается, что Secret `example-com-tls` уже существует в namespace `d8-monitoring`.

Имена для `Ingress` `my-prometheus-api` и Secret `my-basic-auth-secret` указаны для примера. Замените их на более подходящие для вас.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-prometheus-api
  namespace: d8-monitoring
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: my-basic-auth-secret
    nginx.ingress.kubernetes.io/app-root: /graph
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_ssl_certificate /etc/nginx/ssl/client.crt;
      proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
      proxy_ssl_protocols TLSv1.2;
      proxy_ssl_session_reuse on;
spec:
  ingressClassName: nginx
  rules:
    - host: prometheus-api.example.com
      http:
        paths:
          - backend:
              service:
                name: prometheus
                port:
                  name: https
              path: /
              pathType: ImplementationSpecific
      tls:
        - hosts:
            - prometheus-api.example.com
          secretName: example-com-tls
---
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-auth-secret
  namespace: d8-monitoring
type: Opaque
data:
  # Строка basic-auth хешируется с помощью htpasswd.
  auth: Zm9vOiRhchIxJE9GRzNYeWJwJGNrTDBGSErBa29YWUlsSDkuY3lzVDAK # foo:bar

```

Добавьте data source в Grafana:

i В качестве URL необходимо указать `https://prometheus-api.<домен-вашего-кластера>**.`

The screenshot shows the 'Data Sources / FlantPrometheus' configuration page in the Deckhouse UI. The page has a sidebar on the left with navigation icons. The main content area has tabs for 'Settings' and 'Dashboards'. Under 'Settings', there is a 'Name' field set to 'FlantPrometheus' and a 'Default' toggle switch. Below this is the 'HTTP' section with a 'URL' field containing 'https://prometheus-api.<домен-вашего-...>' and a 'Whitelisted Cookies' section with an 'Add Name' button. The 'Auth' section contains several toggle switches: 'Basic auth' (checked), 'With Credentials' (unchecked), 'TLS Client Auth' (unchecked), 'With CA Cert' (unchecked), 'Skip TLS Verify' (unchecked), and 'Forward OAuth Identity' (unchecked). Below the 'Auth' section is the 'Basic Auth Details' section with a 'User' field set to 'foo' and a 'Password' field set to 'configured', with a 'reset' button. At the bottom, there are three more fields: 'Scrape interval' (empty), 'Query timeout' (set to '60s'), and 'HTTP Method' (set to 'Choose' with a dropdown arrow).

- **Basic-авторизация** не является надежной мерой безопасности. Рекомендуется ввести дополнительные меры безопасности, например указать аннотацию `nginx.ingress.kubernetes.io/whitelist-source-range`.
- Из-за необходимости создания Ingress-ресурса в системном пространстве имён подключение таким способом **не рекомендуется**. DKP **не гарантирует** сохранение работоспособности данной схемы подключения в связи с его активными постоянными обновлениями.
- Этот Ingress-ресурс может быть использован для доступа к Prometheus API не только для Grafana, но и для других интеграций, например, для федерации Prometheus.

Подключение стороннего приложения к Prometheus

Подключение к Prometheus защищено с помощью kube-rbac-proxy. Для подключения создайте ServiceAccount с необходимыми правами.

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app:prometheus-access
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["prometheuses/http"]
  resourceNames: ["main", "longterm"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app:prometheus-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app:prometheus-access
subjects:
- kind: ServiceAccount
  name: app
  namespace: default

```

Выполните запрос, используя команду `curl`:

```

apiVersion: batch/v1
kind: Job
metadata:
  name: app-curl
  namespace: default
spec:
  template:
    metadata:
      name: app-curl
    spec:
      serviceAccountName: app
      containers:
      - name: app-curl
        image: curlimages/curl:7.69.1
        command: ["sh", "-c"]
        args:
        - >-
          curl -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" -k -f
          https://prometheus.d8-monitoring:9090/api/v1/query_range?
query=up\&start=1584001500\&end=1584023100\&step=30
          restartPolicy: Never
        backoffLimit: 4

```

`Job` должен завершиться успешно.

Сбор метрик через шлюз (Pushgateway)

Prometheus, находящийся в основе системы мониторинга DKP, преимущественно использует pull-

модель для сбора метрик. При таком подходе происходит опрос экспортеров метрик со стороны DKP. Когда применение pull-модели затруднено, например, для сервисов без постоянного сетевого интерфейса, можно использовать сбор метрик через шлюз (Pushgateway). Pushgateway позволяет таким задачам самим отправлять метрики, которые затем могут быть собраны Prometheus. Важно отметить, что Pushgateway может стать единой точкой отказа и узким местом в системе. Как отправлять метрики из приложения в Pushgateway можно узнать в документации Prometheus.

Пример настройки сбора метрик через шлюз (Pushgateway):

- Включите и настройте модуль `prometheus-pushgateway`.

Включить модуль можно в веб-интерфейсе (Deckhouse Console), или с помощью следующей команды:

```
d8 system module enable prometheus-pushgateway
```

- Укажите названия шлюзов в параметре `instances` модуля `prometheus-pushgateway` через веб-интерфейс, или с помощью следующей команды:

```
d8 k edit mc prometheus-pushgateway
```

Пример конфигурации модуля:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  version: 1
  enabled: true
  settings:
    instances:
      - first
      - second
      - another
```

Адрес экземпляра PushGateway с именем `first` из контейнера пода будет: `http://first.kube-prometheus-pushgateway:9091`.

- Проверьте отправку метрик.

Пример отправки метрики через curl:

```
echo "test_metric{env='dev'} 3.14" | curl --data-binary @- http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp
```

- Проверьте что метрика появилась в системе мониторинга. Она будет доступна через 30 секунд после сбора данных.

Пример PromQL запроса:

```
test_metric{container="prometheus-pushgateway", env="dev", exported_job="myapp",
  instance="10.244.1.155:9091", job="prometheus-pushgateway", pushgateway="prometheus-pushgateway", tier="cluster"} 3.14
```

Удаление метрик из шлюза (Pushgateway)

Пример удаления всех метрик группы `{instance="10.244.1.155:9091", job="myapp"}` через curl:

```
curl -X DELETE http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp/instance/10.244.1.155:9091
```

Обратите внимание, что PushGateway хранит полученные метрики в памяти. При рестарте пода PushGateway все метрики будут утеряны.

Дополнительные ресурсы

- Документация модуля `prometheus-pushgateway`

Настройка системы сбора и хранения метрик

i

Начиная с версии 1.71, в Deckhouse Kubernetes Platform используется Deckhouse Prom++ вместо Prometheus.

Что делает Prometheus?

Prometheus собирает метрики и выполняет правила:

- Для каждого *target* (цели мониторинга) с заданной периодичностью `scrape_interval` Prometheus выполняет HTTP-запрос на этот *target*, получает в ответ метрики в собственном формате и сохраняет их в свою базу данных.
- Каждый `evaluation_interval` обрабатывает правила (*rules*), на основании чего:
 - отправляет алерты;
 - или сохраняет новые метрики (результат выполнения правил) в свою базу данных.

Как работает Prometheus?

Prometheus устанавливается модулем `prometheus` DKP, который выполняет следующие функции:

- определяет следующие кастомные ресурсы:
 - `Prometheus` — определяет инсталляцию (кластер) *Prometheus*.
 - `ServiceMonitor` — определяет, как собирать метрики с сервисов.
 - `Alertmanager` — определяет кластер *Alertmanager*’ов.
 - `PrometheusRule` — определяет список *Prometheus rules*.
- следит за этими ресурсами, а также:
 - генерирует `StatefulSet` с самим *Prometheus*.
 - создает секреты с необходимыми для работы Prometheus конфигурационными файлами (`prometheus.yaml` — конфигурация Prometheus, и `configmaps.json` — конфигурация для `prometheus-config-reloader`).
 - следит за ресурсами `ServiceMonitor` и `PrometheusRule` и на их основании обновляет конфигурационные файлы *Prometheus* через внесение изменений в секреты.

Включить модуль можно с использованием следующего ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings:
    auth:
      password: xxxxxx
      retentionDays: 7
      storageClass: rbd
      nodeSelector:
        node-role/monitoring: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: monitoring
```

Полное описание всех настроек доступно в документации модуля `prometheus`.

Дополнительные ресурсы

- Документация модуля `prometheus`

Мониторинг SLA кластера

DKP может собирать статистику о доступности компонентов кластера и компонентов самого DKP. Эти данные позволяют оценить степень выполнения SLA, а также получить информацию о доступности в веб-интерфейсе.

Кроме того, с помощью кастомного ресурса UpmeterRemoteWrite можно экспортировать метрики доступности по протоколу Prometheus Remote Write.

Чтобы начать собирать метрики доступности и активировать интерфейс, включите модуль `upmeter` в веб-интерфейсе Deckhouse или воспользуйтесь командой:

```
d8 system module enable upmeter
```

Настройка модуля

Модуль `upmeter` настраивается с помощью `ModuleConfig upmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  version: 3
  enabled: true
  settings:
```

Перечень всех настроек доступен в документации модуля.

Интерфейс

DKP предоставляет два веб-интерфейса для оценки доступности:

- Страница статуса.

Получить адрес страницы можно в веб-интерфейсе на главной странице в разделе «Инструменты» (блок «Статус-страница»), или выполнив команду:

```
d8 k -n d8-upmeter get ing status -o jsonpath='{.spec.rules[*].host}'
```

Пример веб-интерфейса страницы статуса:

Status as of 1:37:27 PM	
Control plane	Operational
The availability of Kubernetes control-plane	
Deckhouse	Operational
The availability of deckhouse and working hook	
Extensions	Operational
The availability of extensions apps	
Load Balancing	Operational
The availability of traffic load balancing and its configuration controllers	
Monitoring and Autoscaling	Operational
The availability of monitoring and autoscaling applications in the cluster	
Nginx	Operational
The availability of Nginx Ingress Controllers	
Node Groups	Operational
The availability of CloudEphemeral nodes	
Synthetic	Operational
The availability of sample application, and network connectivity	

- Страница доступности компонентов.

Получить адрес страницы можно в веб-интерфейсе на главной странице в разделе «Инструменты» (блок «Доступность компонентов»), или выполнив команду:

```
d8 k -n d8-upmeter get ing webui -o jsonpath='{.spec.rules[*].host}'
```

Пример страницы доступности компонентов:



Экспорт метрик статуса

Пример конфигурации UpmeterRemoteWrite для экспорта метрик статуса по протоколу Prometheus Remote Write:

```

apiVersion: deckhouse.io/v1
kind: UpmeterRemoteWrite
metadata:
  labels:
    heritage: upmeter
    module: upmeter
    name: victoriametrics
spec:
  additionalLabels:
    cluster: cluster-name
    some: fun
  config:
    url: https://upmeter-victoriametrics.whatever/api/v1/write
    basicAuth:
      password: "Cdp#Cd.0xfZsx4*89SZ"
      username: upmeter
    intervalSeconds: 300

```

Аутентификация

По умолчанию для аутентификации используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication`. Если эти варианты отключены, модуль включит базовую аутентификацию со сгенерированным паролем.

Посмотреть сгенерированный пароль можно с помощью команды:

```

d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.webui.password'

```

Чтобы сгенерировать новый пароль, нужно удалить Secret:

```

d8 k -n d8-upmeter delete secret/basic-auth-webui

```

Посмотреть сгенерированный пароль для страницы статуса можно с помощью команды:

```

d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.status.password'

```

Чтобы сгенерировать новый пароль для страницы статуса, нужно удалить секрет:

```

d8 k -n d8-upmeter delete secret/basic-auth-status

```

i **Внимание.** Параметры `auth.status.password` и `auth.webui.password` больше не поддерживаются.

Особенности работы подов upmeter

Тесты `upmeter` создают временные поды для проверки работоспособности компонентов Kubernetes. Поэтому некоторые поды периодически удаляются, остаются в состоянии `Pending` или перемещаются между узлами.

Ниже описано, какие объекты участвуют в проверках:

- `upmeter-probe-scheduler` — проверка планировщика. Тест создает под, размещает его на узле и затем удаляет.
- `upmeter-probe-controller-manager` — проверка `kube-controller-manager`. Тест создает `StatefulSet` и убеждается, что `StatefulSet` создал под. Размещение пода на узле в этом тесте не проверяется, поэтому создается под, который заведомо не может быть размещен и остаётся в состоянии `Pending`. Затем `StatefulSet` удаляется, и проверяется, что созданный им под тоже удален.
- `smoke-mini` — проверка сетевой связности между узлами. Создаются пять `StatefulSet` с одной репликой каждый. Тест проверяет связность между подами `smoke-mini` и подами

`upmeter-agent` на master-узлах. Раз в минуту один из подов `smoke-mini` переносится на другой узел.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `upmeter`
- Документация модуля `user-authn`

Настройка интеграций

Перенаправление алертов в Zabbix

Deckhouse Kubernetes Platform поддерживает интеграцию с системой мониторинга Zabbix. Для этого используется внешний скрипт, который получает алерты из Deckhouse через `kubectl` и передаёт их в Zabbix с помощью Zabbix-агента.

Для работы скрипта потребуются:

- установленный и настроенный zabbix-agent;
- утилиты `bash`, `jq`;
- `kubectl` с рабочим конфигурационным файлом и правами на выполнение команды `kubectl get clusteralerts`.

Установка

1. Импортируйте шаблон в Zabbix:
 - В веб-интерфейсе Zabbix перейдите в раздел «Data collection → Templates»
 - Нажмите «Import» и загрузите файл `zbx_export_templates.yaml`:

```
zabbix_export:
  version: '6.4'
  template_groups:
    - uuid: 7df96b18c230490a9a0a9e2307226338
      name: Templates
  templates:
    - uuid: 91d9cf9d023749a3997bc6429ef45fe1
      template: 'Module d8alerts'
      name: 'Deckhouse Alerts'
      groups:
        - name: Templates
      discovery_rules:
        - uuid: e9be18e9305e48e887c43e1ad9ea9c31
          name: 'Deckhouse Alert Discovery'
          key: d8alerts.discovery
          lifetime: 3d
          item_prototypes:
            - uuid: c4152d379d7a46cab23f7437cdb632c1
              name: 'Deckhouse alert: {#ALERTNAME} severity'
              key: 'd8alerts.severity[{#ALERTID}]'
              history: 1d
              trends: '0'
              description: |
                Summary: {#SUMMARY}

                {#DESCRIPTION}

                Labels: {#LABELS}
                To get more information: `kubectl describe clusteralert
                {#ALERTID}`
              preprocessing:
                - type: REGEX
                  parameters:
                    - (\d+)
                    - \1
```



```

tags:
  - tag: Application
    value: deckhouse
trigger_prototypes:
  - uuid: b27f8d03dc4c4d33bab4193a6ce57ae7
    expression: 'last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) <= 2 and last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) > 0'
    name: '{{#ALERTNAME}} is firing'
    priority: DISASTER
    description: |
      Summary: {{#SUMMARY}}

      {{#DESCRIPTION}}

      Labels: {{#LABELS}}
      To get more information: `kubectl describe clusteralert
{{#ALERTID}}`

  - uuid: d1935044cf3f49df8bc53f738e36b683
    expression: 'last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) >= 3 and last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) <= 4'
    name: '{{#ALERTNAME}} is firing'
    priority: HIGH
    description: |
      Summary: {{#SUMMARY}}

      {{#DESCRIPTION}}

      Labels: {{#LABELS}}
      To get more information: `kubectl describe clusteralert
{{#ALERTID}}`

  - uuid: 0b134c599d9d4de79f906fbd8e749ec2
    expression: 'last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) >= 5 and last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) <= 6'
    name: '{{#ALERTNAME}} is firing'
    priority: AVERAGE
    description: |
      Summary: {{#SUMMARY}}

      {{#DESCRIPTION}}

      Labels: {{#LABELS}}
      To get more information: `kubectl describe clusteralert
{{#ALERTID}}`

  - uuid: 6967dc80d6414239b2d597447815048a
    expression: 'last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) >= 7 and last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) <= 8'
    name: '{{#ALERTNAME}} is firing'
    priority: WARNING
    description: |
      Summary: {{#SUMMARY}}

      {{#DESCRIPTION}}

      Labels: {{#LABELS}}
      To get more information: `kubectl describe clusteralert
{{#ALERTID}}`

  - uuid: 8e531a2e6a0f47849c1dffa9a1733e1
    expression: 'last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) >= 9 and last(/Module
d8alerts/d8alerts.severity[{{#ALERTID}}]) <= 10'

```

```

        name: '#{ALERTNAME} is firing'
        priority: INFO
        description: |
            Summary: #{SUMMARY}

            #{DESCRIPTION}

        Labels: #{LABELS}
        To get more information: `kubectl describe clusteralert
        #{ALERTID}`

```

- После установки шаблона укажите ему группу, соответствующую агентам, с которых будет происходить сбор метрик.

2. Настройте Zabbix-агент:

- Скопируйте файл `d8alerts.conf` в директорию, указанную в параметре `Include` основного конфига Zabbix-агента (обычно расположен по пути `/etc/zabbix/zabbix_agentd.d/`):

```

# LLD of deckhouse cluster alerts
UserParameter=d8alerts.discovery,/etc/zabbix/scripts/clusteralerts.sh
discovery

# Severity of a specific alert by its ID
UserParameter=d8alerts.severity[*],/etc/zabbix/scripts/clusteralerts.sh
severity "$1"

```

- Скопируйте скрипт `clusteralerts.sh` в директорию `/etc/zabbix/scripts/`:

```

#!/bin/bash

MODE="$1"
ALERT_ID="$2"

get_alerts_json() {
    kubectl get clusteralerts -o json
}

get_alert_by_id() {
    local id="$1"
    kubectl get clusteralerts "$id" -o json
}

if [[ "$MODE" == "discovery" ]]; then
    get_alerts_json | jq -c '{
        data: [.items[] | {
            "#{ALERTID}": .metadata.name,
            "#{ALERTNAME}": .alert.name,
            "#{SEVERITY}": (.alert.severityLevel | tonumber),
            "#{DESCRIPTION}": .alert.description,
            "#{SUMMARY}": .alert.summary,
            "#{LABELS}": (.alert.labels | to_entries | map("\(.key)=\(.value)") |
join(", "))
        }}
    }'
elif [[ "$MODE" == "severity" && -n "$ALERT_ID" ]]; then
    get_alert_by_id "$ALERT_ID" | jq -r '.alert.severityLevel'
else
    echo "Invalid usage"
    exit 1
fi

```

- Убедитесь, что скрипт имеет права на выполнение:

```
chmod +x /etc/zabbix/scripts/clusteralerts.sh
```

3. Проверьте доступ:

- Убедитесь, что скрипт имеет доступ к кластеру и может получать информацию об алертах:

```
/etc/zabbix/scripts/clusteralerts.sh discovery
```

Вывод должен содержать список алертов с их статусами и уровнями критичности.

4. Перезапустите Zabbix-агент для применения изменений:

```
systemctl restart zabbix-agent
```

Устранение неполадок

1. Проверьте логи Zabbix-агента:

```
tail -f /var/log/zabbix/zabbix_agentd.log
```

2. Проверьте работу скрипта:

```
/etc/zabbix/scripts/clusteralerts.sh discovery
/etc/zabbix/scripts/clusteralerts.sh severity "ID_АЛЕРТА"
```

Убедитесь, что скрипт корректно выполняется и возвращает ожидаемые данные.

3. Проверьте права пользователя `zabbix`. Убедитесь, что агент выполняет скрипт от пользователя с нужными правами доступа к кластеру:

- Скрипт запускается от имени пользователя `zabbix`:

```
sudo -u zabbix /etc/zabbix/scripts/clusteralerts.sh
```

- Переменная `KUBECONFIG` доступна пользователю:

Если конфигурационный файл Kubernetes недоступен по умолчанию, укажите его явно. Для этого сохраните `kubeconfig`, например, в `/etc/zabbix/kubeconfig` и добавьте в конфигурацию агента:

```
UserParameter=d8alerts.discovery,export
KUBECONFIG=/etc/zabbix/kubeconfig;/etc/zabbix/scripts/clusteralerts.sh discovery
```

Отправка алертов в Telegram

Alertmanager поддерживает прямую отправку алертов в Telegram.

Создайте Secret в пространстве имен `d8-monitoring`:

```
apiVersion: v1
kind: Secret
metadata:
  name: telegram-bot-secret
  namespace: d8-monitoring
stringData:
  token: "562696849:AAExcuJ8H6z4pTlPuocbrXXXXXXXXXXXXX"
```

Разверните кастомный ресурс CustomAlertManager:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: telegram
spec:
  type: Internal
  internal:
    receivers:
      - name: telegram
        telegramConfigs:
          - botToken:
              name: telegram-bot-secret
              key: token
              chatID: -30490XXXXX
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: telegram
      repeatInterval: 12h
```

Поля `token` в `Secret`е и `chatID` в ресурсе `CustomAlertmanager` необходимо поставить свои. Подробнее о Telegram API.

Пример отправки алертов в Slack с фильтром

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: slack
spec:
  internal:
    receivers:
      - name: devnull
      - name: slack
      slackConfigs:
        - apiURL:
            key: apiURL
            name: slack-apiurl
            channel: '{{ dig .Values.werf.env .Values.slack.channel._default
.Values.slack.channel }}'
            fields:
              - short: true
                title: Severity
                value: '{{`{{ .CommonLabels.severity_level }}`}}'
              - short: true
                title: Status
                value: '{{`{{ .Status }}`}}'
              - title: Summary
                value: '{{`{{ range .Alerts }}`}}`{{`{{ .Annotations.summary }}`}}`{{`{{ end
}}`}}`'
              - title: Description
                value: '{{`{{ range .Alerts }}`}}`{{`{{ .Annotations.description }}`}}`{{`{{
end }}`}}`'
              - title: Labels
                value: '{{`{{ range .Alerts }}`}}`{{`{{ range .Labels.SortedPairs }}`}}`{{`{{
printf "%s:
          %s\n" .Name .Value }}`}}`{{`{{ end }}`}}`{{`{{ end }}`}}`'
              - title: Links
                value: '{{`{{ (index .Alerts 0).GeneratorURL }}`}}`'
                title: '{{`{{ .CommonLabels.alertname }}`}}`'
    route:
      groupBy:
        - '...'
      receiver: devnull
      routes:
        - matchers:
            - matchType: =~
              name: severity_level
              value: "^[4-9]$"
              receiver: slack
        repeatInterval: 12h
      type: Internal

```

Пример отправки алертов в Opsgenie

```
- name: opsgenie
  opsgenieConfigs:
    - apiKey:
        key: data
        name: opsgenie
      description: |
        {{ range .Alerts }}{{ .Annotations.summary }} {{ end }}
        {{ range .Alerts }}{{ .Annotations.description }} {{ end }}
      message: '{{ .CommonLabels.alertname }}'
      priority: P1
      responders:
        - id: team_id
          type: team
```

Пример отправки алертов по электронной почте

Создайте секрет с паролем от аккаунта электронной почты. Пароль, закодированный в формате Base64, укажите в поле `password`:

```
apiVersion: v1
kind: Secret
metadata:
  name: am-mail-server-pass
  namespace: d8-monitoring
data:
  password: BASE64_ENCODED_PASSWORD_HERE
```

Измените значения в примере ресурса `CustomAlertManager` в соответствии с актуальными для вашей инфраструктуры значениями и примените его:

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: mail
spec:
  type: Internal
  internal:
    receivers:
      - name: devnull
      - name: mail
    emailConfigs:
      - to: oncall@example.com
        from: prom@example.com
        smarthost: mx.example.com:587
        authIdentity: prom@example.com
        authUsername: prom@example.com
        authPassword:
          key: password
          name: am-mail-server-pass
        # Если вы используете custom CA на сервере, можете поместить публичную
        часть CA в ConfigMap в пространстве имен d8-monitoring
        # tlsConfig:
        #   insecureSkipVerify: true
        #   ca:
        #     configMap:
        #       key: ca.pem
        #       name: alertmanager-mail-server-ca
        sendResolved: true
        requireTLS: true
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: devnull
      repeatInterval: 24h
      routes:
        - matchers:
            - matchType: =~
              name: severity_level
              value: "^[1-4]$"
          receiver: mail

```

Дополнительные ресурсы

- Кастомные ресурсы модуля `prometheus`

Логирование

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства для сбора, доставки и кратковременного хранения логов.

Возможности DKP в области логирования:

- сбор логов с подов и узлов кластера;
- обработка логов, включая обогащение метаданными и фильтрацию сообщений;
- доставка в различные системы хранения и анализа, включая Loki, Elasticsearch, Splunk и другие;
- кратковременное хранение логов в кластере с возможностью поиска и визуализации через Grafana.

В следующих разделах описано, как настроить сбор и доставку логов, а также организовать их кратковременное хранение в пределах кластера.

Сбор и доставка логов

В Deckhouse Kubernetes Platform (DKP) предусмотрен сбор и доставка логов из узлов и подов кластера во внутреннюю или внешние системы хранения.

DKP позволяет:

- собирать логи из всех или отдельных подов и пространств имён;
- фильтровать логи по лейблам, содержимому сообщений и другим признакам;
- направлять логи одновременно в несколько хранилищ (например, Loki и Elasticsearch);
- обогащать логи метаданными Kubernetes;
- использовать буферизацию логов для повышения производительности.

Общий механизм сбора, доставки и фильтрации логов подробно описан в разделе «Архитектура».

Администраторам DKP доступна настройка сбора и отправки логов с помощью трёх кастомных ресурсов:

- `ClusterLoggingConfig` — описывает источник логов на уровне кластера, включая правила сбора, фильтрации и парсинга;
- `PodLoggingConfig` — описывает источник логов в рамках заданного пространства имён, включая правила сбора, фильтрации и парсинга;
- `ClusterLogDestination` — задаёт параметры хранилища логов.

На основе этих ресурсов формируется *pipeline*, который используется в DKP для чтения логов и дальнейшей работы с ними с помощью модуля `log-shipper`. Полный перечень настроек модуля `log-shipper` доступен в отдельном разделе документации.

Настройка сбора и доставки логов

Ниже приведён вариант базовой конфигурации DKP, при котором логи со всех подов кластера отправляются в хранилище на базе Elasticsearch.

Для настройки выполните следующие шаги:

1. Включите модуль `log-shipper` с помощью следующей команды:

```
d8 system module enable log-shipper
```

2. Создайте ресурс `ClusterLoggingConfig`, который задаёт правила сбора логов. Данный ресурс позволяет вам настроить сбор логов с подов в определенном пространстве имён и с определенным лейблом, гибко настраивать парсинг многострочных логов и задавать другие правила.

В этом примере указывается, что нужно собирать логи со всех подов и отправлять их в Elasticsearch:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
    - es-storage
```

3. Создайте ресурс `ClusterLogDestination`, который описывает параметры отправки логов в хранилище. Данный ресурс позволяет вам указать одно или несколько хранилищ и описать параметры подключения, буферизации и дополнительные лейблы, которые будут применяться к логам перед отправкой.

В этом примере в качестве принимающего хранилища указан Elasticsearch:


```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==

```

Интеграция с внешними системами

Вы можете настроить DKP на работу с внешними системами хранения и анализа логов, такими как Elasticsearch, Splunk, Logstash и другими, используя параметр `type` ресурса `ClusterLogDestination`.

Elasticsearch

Чтобы отправлять логи в Elasticsearch, создайте ресурс `ClusterLogDestination`, следуя этому примеру:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==

```

Использование шаблона индексов

Чтобы отправлять сообщения в определенные индексы на основе метаданных с помощью шаблонов индексов, используйте следующую конфигурацию:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s--%F"

```

В приведенном примере для каждого пространства имён Kubernetes будет создан свой индекс в Elasticsearch.

Эта функция удобна в комбинации с параметром `extraLabels`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s---%F"
  extraLabels:
    service: ""

```

- Если сообщение имеет формат JSON, поле `service_name` этого документа JSON перемещается на уровень метаданных.
- Новое поле метаданных `service` используется в шаблоне индекса.

Поддержка Elasticsearch < 6.X

Для работы с версиями Elasticsearch ранее 6.0 включите поддержку индексов `docType` с помощью ресурса `ClusterLogDestination`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    docType: "myDocType" # Укажите значение здесь. Оно не должно начинаться с '_'.
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCG==

```

Splunk

Чтобы настроить отправку событий в Splunk, выполните следующие шаги:

1. Настройте Splunk:
 - Определите endpoint. Он должен совпадать с именем вашего экземпляра Splunk с портом 8088, но без указания пути, например, `https://prd-p-xxxxxx.splunkcloud.com:8088`.
 - Создайте токен для доступа. Для этого в Splunk откройте раздел **Setting -> Data inputs**, добавьте новый **HTTP Event Collector** и скопируйте сгенерированный токен.
 - Укажите индекс Splunk для хранения логов, например, `logs`.
2. Настройте DKP, добавив ресурс `ClusterLogDestination` для отправки логов в Splunk:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: splunk
spec:
  type: Splunk
  splunk:
    endpoint: https://prd-p-xxxxxx.splunkcloud.com:8088
    token: xxxx-xxxx-xxxx
    index: logs
    tls:
      verifyCertificate: false
      verifyHostname: false

```

i `destination` не поддерживает лейблы пода для индексирования. Чтобы добавить нужные лейблы, используйте опцию `extraLabels`:

```
extraLabels:
  pod_label_app: ''
```

Logstash

Чтобы настроить отправку логов в Logstash, выполните следующее:

1. Настройте входящий поток `tcp` с кодеком `json` на стороне Logstash.

Пример конфигурации Logstash:

```
input {
  tcp {
    port => 12345
    codec => json
  }
}
output {
  stdout { codec => json }
}
```

2. Добавьте ресурс `ClusterLogDestination`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: logstash
spec:
  type: Logstash
  logstash:
    endpoint: logstash.default:12345
```

Graylog

Чтобы настроить отправку логов в Graylog, выполните следующее:

1. Убедитесь, что в Graylog настроен входящий поток для приема сообщений по протоколу TCP на указанном порте.
2. Создайте ресурс `ClusterLogDestination`, следуя примеру:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: test-socket2-dest
spec:
  type: Socket
  socket:
    address: graylog.svc.cluster.local:9200
    mode: TCP
    encoding:
      codec: GELF
```

Формат сообщений

Вы можете выбрать формат отправляемых сообщений, используя параметр `.encoding.codec` ресурса `ClusterLogDestination`:

- CEF

- GELF
- JSON
- Syslog
- Text

Ниже примеры конфигурации для некоторых из них.

Syslog

Используйте следующий пример конфигурации для отправки сообщений через сокет по протоколу TCP в формате syslog:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: rsyslog
spec:
  type: Socket
  socket:
    mode: TCP
    address: 192.168.0.1:3000
    encoding:
      codec: Syslog
  extraLabels:
    syslog.severity: "alert"
    # Поле request_id должно присутствовать в сообщении.
    syslog.message_id: ""
```

CEF

DKP может отправлять логи в формате CEF через использование `codec: CEF`, с переопределением `cef.name` и `cef.severity` по значениям из поля `message` лога приложения в формате JSON.

В примере ниже `app` и `log_level` — это ключи, содержащие значения для переопределения:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: siem-kafka
spec:
  extraLabels:
    cef.name: ''
    cef.severity: ''
  type: Kafka
  kafka:
    bootstrapServers:
      - my-cluster-kafka-brokers.kafka:9092
    encoding:
      codec: CEF
    tls:
      verifyCertificate: false
      verifyHostname: true
    topic: logs
```

Также можно задать значения вручную:

```
extraLabels:
  cef.name: 'TestName'
  cef.severity: '1'
```

Преобразование логов

Вы можете настроить один или несколько видов трансформаций, которые будут применяться к логам перед отправкой в хранилище.

Преобразование записи в структурированный объект

Трансформация `ParseMessage` позволяет преобразовать строку в поле `message` в структурированный JSON-объект на основе одного или нескольких заданных форматов (`String`, `Klog`, `SysLog` и другие).



При использовании нескольких трансформаций `ParseMessage` преобразование строки (`sourceFormat: String`) должно выполняться в последнюю очередь.

Пример настройки преобразования записей смешанных форматов:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
  - action: ParseMessage
    parseMessage:
      sourceFormat: JSON
  - action: ParseMessage
    parseMessage:
      sourceFormat: Klog
  - action: ParseMessage
    parseMessage:
      sourceFormat: String
      string:
        targetField: "text"
```

Пример изначальной записи в логге:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
{"level" : { "severity": "info" }, "msg" : "fetching.module.release"}
I0505 17:59:40.692994 28133 klog.go:70] hello from klog
```

Результат преобразования:

```
{... "message": {
  "text": "/docker-entrypoint.sh: Configuration complete; ready for start up"
}}
{... "message": {
  "level" : "{ \"severity\": \"info\" }",
  "msg" : "fetching.module.release"
}}
{... "message": {
  "file": "klog.go",
  "id": 28133,
  "level": "info",
  "line": 70,
  "message": "hello from klog",
  "timestamp": "2025-05-05T17:59:40.692994Z"
}}
```

Замена лейблов

Трансформация `ReplaceKeys` позволяет рекурсивно заменить все совпадения шаблона `source` на значение `target` в указанных ключах лейблов.



Перед применением трансформации `ReplaceKeys` к полю `message` или его вложенным полям преобразуйте запись лога в структурированный объект с помощью трансформации `ParseMessage`.

Пример настройки замены точек на нижние подчеркивания в лейблах:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: replace-dot
spec:
  ...
  transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
```

Пример изначальной записи в логе:

```
{"msg" : "fetching.module.release"} # Лейбл пода pod.app=test
```

Результат преобразования:

```
{... "message": {
  "msg" : "fetching.module.release"
},
  "pod_labels": {
    "pod_app": "test"
  }
}
```

Удаление лейблов

Трансформация `DropLabels` позволяет удалить указанные лейблы из структурированного JSON-сообщения.



Перед применением трансформации `DropLabels` к полю `message` или его вложенным полям преобразуйте запись лога в структурированный объект с помощью трансформации `ParseMessage`.

Пример конфигурации с удалением лейбла и предварительной трансформацией `ParseMessage`:

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: DropLabels
      dropLabels:
        labels:
          - .message.example

```

Пример изначальной записи в логе:

```

{"msg" : "fetching.module.release", "example": "test"}

```

Результат преобразования:

```

{... "message": {
  "msg" : "fetching.module.release"
}}

```

Фильтрация логов

В DKP предусмотрены фильтры, позволяющие исключить лишние сообщения для оптимизации процесса сбора логов:

- `labelFilter` — применяется к метаданным, например, к имени контейнера (`container`), пространству имён (`namespace`) или имени пода (`pod_name`);
- `logFilter` — применяется к полям сообщения, если оно в JSON-формате.

Сборка логов из определенного контейнера

Чтобы настроить фильтрацию с помощью `labelFilter`, создайте ресурс `ClusterLoggingConfig`, используя конфигурацию ниже в качестве примера.

В этом случае фильтр отбирает логи из контейнеров с именем `nginx`, а затем отправляет их во внутреннее хранилище на базе Loki.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: nginx-logs
spec:
  type: KubernetesPods
  labelFilter:
    - field: container
      operator: In
      values: [nginx]
  destinationRefs:
    - loki-storage

```

Сборка логов без заданной строки

Пример конфигурации для сбора логов с фильтрацией через `labelFilter`, где оператор `NotRegex` исключает строки, соответствующие заданному регулярному выражению.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
    - loki-storage
  labelFilter:
    - field: message
      operator: NotRegex
      values:
        - .*GET /status" 200$

```

Аудит событий kubelet

Пример конфигурации для сбора и фильтрации событий аудита, связанных с работой kubelet, хранящихся в файле `/var/log/kube-audit/audit.log`. Фильтрация выполняется с помощью `logFilter`, который ищет в поле `userAgent` записи, соответствующие регулярному выражению `"kubelet.*"`.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: ["kubelet.*"]
  destinationRefs:
    - loki-storage

```

Системные логи DKP

Пример конфигурации для сбора системных логов DKP, находящихся в файле `/var/log/syslog`. Фильтрация сообщений с помощью `labelFilter` позволяет выделить только те записи, которые относятся к следующим компонентам: `d8-kubelet-forker`, `containerd`, `bashible` и `kernel`.


```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: system-logs
spec:
  type: File
  file:
    include:
      - /var/log/syslog
  labelFilter:
    - field: message
      operator: Regex
      values:
        - .*d8-kubelet-forker.*
        - .*containerd.*
        - .*bashible.*
        - .*kernel.*
  destinationRefs:
    - loki-storage

```

i Если вам нужны логи только одного пода или небольшой группы подов, используйте `kubernetesPods`, чтобы ограничить область сбора. Фильтры следует применять только для тонкой настройки.

Буферизация логов

Использование буферизации повышает надежность и производительность системы сбора логов. Буферизация может быть полезна в следующих случаях:

- **Временные перебои с подключением.** Если есть временные перебои или нестабильность соединения с системой хранения логов (например, с Elasticsearch), буфер позволяет временно сохранять логи и отправить их, когда соединение восстановится.
- **Сглаживание пиков нагрузки.** При внезапных всплесках объема логов буфер позволяет сгладить пиковую нагрузку на систему хранения, предотвращая её перегрузку и потенциальную потерю данных.
- **Оптимизация производительности.** Буферизация помогает оптимизировать производительность системы сбора логов за счёт накопления логов и отправки их группами, что снижает количество сетевых запросов и улучшает общую пропускную способность.

За настройку буферизации отвечает параметр `buffer` ресурса `ClusterLogDestination`.

Пример включения буферизации в оперативной памяти

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    memory:
      maxEvents: 4096
    type: Memory
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Пример включения буферизации на диске

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
      type: Disk
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Пример определения поведения при переполнении буфера

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
      type: Disk
      whenFull: DropNewest
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Отладка и расширенные возможности

Включение debug-логов агента log-shipper

Чтобы включить debug-логи агента `log-shipper` на узлах с информацией об HTTP-запросах, переиспользовании подключения, трассировке и прочими данными, включите параметр `debug` в конфигурации модуля `log-shipper`.

Пример конфигурации модуля:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 1
  enabled: true
  settings:
    debug: true

```

Дополнительная информация о каналах передачи логов

Используя команды для Vector, можно получить дополнительную информацию о каналах передачи данных.

Для начала подключитесь к одному из подов `log-shipper`:

```

d8 k -n d8-log-shipper get pods -O wide | grep $node
d8 k -n d8-log-shipper exec $pod -it -c vector -- bash

```

Выполняйте следующие команды из командной оболочки пода.

Обзор топологии

Чтобы получить схему топологии вашей конфигурации:

1. Выполните команду `vector graph`. Будет сформирована схема в формате DOT.
2. Используйте WebGraphviz или аналогичный сервис для отрисовки схемы на основе содержимого DOT-файла.

Пример схемы для одного канала передачи логов в формате ASCII:

```
+-----+
| d8_cluster_source_flant-integration-d8-logs |
+-----+
|
| v
+-----+
|      d8_tf_flant-integration-d8-logs_0      |
+-----+
|
| v
+-----+
|      d8_tf_flant-integration-d8-logs_1      |
+-----+
|
| v
+-----+
| d8_cluster_sink_flant-integration-loki-storage |
+-----+
```

Мониторинг нагрузки на каналы

Чтобы посмотреть объем трафика на каждом этапе обработки логов, используйте команду `vector top`.

Пример вывода команды:

Vector							
http://127.0.0.1:8686/graphql Sampling @ 500ms							
Components							
ID	Output	Kind	Type	Events In	Events Out	Bytes	Errors
d8_cluster_sink_flant	--	sink	loki	324,198 (--/s)	324,198 (--/s)	N/A	--
d8_cluster_source fla	--	source	kubernetes_log	324,198 (--/s)	324,198 (--/s)	1,099,955,306	--
d8_tf_flant-integrati	--	transform	remap	324,198 (--/s)	324,198 (--/s)	N/A	--
d8_tf_flant-integrati	--	transform	remap	324,198 (--/s)	324,198 (--/s)	N/A	--
internal_metrics	--	source	internal_metri	5,158,622 (--/s)	5,158,622 (--/s)	N/A	--
prometheus_sink	--	sink	prometheus	5,158,622 (--/s)	N/A	N/A	--

Получение необработанных и промежуточных логов

Для просмотра входных данных на разных стадиях обработки логов используйте команду `vector tap`. Указав в ней ID конкретного этапа обработки, вы сможете увидеть логи которые поступают на этом этапе. Также поддерживаются выборки в формате glob, например, `cluster_logging_config/*`.

Примеры:

- Просмотр логов до применения правил трансформаций (`cluster_logging_config/*` является первой стадией обработки согласно выводу команды `vector graph`):

```
vector tap 'cluster_logging_config/*'
```

- Изменённые логи, поступающие на вход следующих в цепочке компонентов каналов:

```
vector tap 'transform/*'
```

Отладка VRL-правил

Для отладки правил на языке Vector Remap Language (VRL) используйте команду `vector vrl`.

Пример VRL-программы:

```
. = {"test1": "lynx", "test2": "fox"}
del(.test2)
.
```

Добавление поддержки нового source или sink

Модуль `log-shipper` в DKP собирается на основе Vector с ограниченным набором cargo-функций, чтобы минимизировать размер запускаемого файла и ускорить сборку.

Чтобы посмотреть весь список поддерживаемых функций, выполните команду `vector list`.

Если нужный source или sink отсутствует, добавьте соответствующую cargo-функцию в Dockerfile.

Особые случаи

Сбор логов с продуктивных пространств имён через опцию `labelSelector`

Если в вашем кластере пространства имён размечены с помощью лейблов (например, `environment=production`), вы можете использовать опцию `labelSelector` для сбора логов из продуктивных пространств имён.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: production-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchLabels:
          environment: production
  destinationRefs:
    - loki-storage
```

Лейбл для исключения подов и пространств имён

В DKP предусмотрен лейбл `log-shipper.deckhouse.io/exclude=true` для исключения определенных подов и пространств имён. Он помогает остановить сбор логов с подов и пространств имён без изменения глобальной конфигурации.

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    log-shipper.deckhouse.io/exclude: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
spec:
  ...
  template:
    metadata:
      labels:
        log-shipper.deckhouse.io/exclude: "true"

```

Дополнительные ресурсы

- Документация модуля `log-shipper`

Кратковременное хранение логов

Deckhouse Kubernetes Platform (DKP) предоставляет встроенное решение для кратковременного хранения логов на базе проекта Grafana Loki.

Хранилище разворачивается в кластере и интегрируется с системой сбора логов. После настройки ресурсов `ClusterLoggingConfig`, `PodLoggingConfig` и `ClusterLogDestination` логи автоматически поступают со всех системных компонентов. Настроенное хранилище добавляется в Grafana в качестве источника данных для визуализации и анализа.

Сбор логов с пользовательских приложений настраивается отдельно.

Параметры кратковременного хранилища задаются в настройках модуля `loki`. В том числе возможно настроить размер диска и срок хранения, задать используемый `StorageClass` и ресурсы.



Кратковременное хранилище на базе Grafana Loki не поддерживает работу в режиме высокой доступности. Для долговременного хранения важных логов используйте внешнее хранилище.

Интеграция с Grafana Cloud

Чтобы настроить работу DKP с платформой Grafana Cloud, выполните следующие шаги:

1. Создайте ключ доступа к API Grafana Cloud.
2. Закодируйте токен доступа к Grafana Cloud в формате Base64:

URL: `https://logs-prod-eu-west-0.grafana.net`
Basic Auth: ☒ (checked)
User:
Password: Your Grafana.com API Key. [Generate now](#)

```
echo -n "<YOUR-GRAFANACLOUD-TOKEN>" | base64 -w0
```

3. Создайте ресурс `ClusterLogDestination`, следуя примеру:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  loki:
    auth:
      password: PFlPVVItR1JBRkF0QUNMT1VELVRPS0V0Pg==
      strategy: Basic
      user: "<YOUR-GRAFANACLOUD-USERNAME>"
      endpoint: <YOUR-GRAFANACLOUD-URL> # Например https://logs-prod-us-central1.grafana.net или https://logs-prod-eu-west-0.grafana.net
      type: Loki

```

Переход с Grafana Promtail

Для миграции с Promtail отредактируйте URL-адрес Loki, убрав из него путь `/loki/api/v1/push`.

Агент логирования Vector, который используется в DKP, автоматически добавит этот путь при отправке данных в Loki.

Дополнительные ресурсы

- Кастомные ресурсы модуля `log-shipper`
- Параметры модуля `loki`

Мультиотенантность

Мультиотенантность — это возможность создавать в кластере Kubernetes изолированные окружения (проекты). Проекты похожи на неймспейсы (Namespace), но имеют больше возможностей. Неймспейсы используются только для логического разделения ресурсов в Kubernetes, и, например, не ограничивают сетевое взаимодействие, потребление ресурсов подами и возможности монтирования директории хоста. Такие особенности неймспейсов не полностью соответствуют современным требованиям к разработке. По умолчанию для неймспейсов также не включена сборка логов, аудит и сканирование на уязвимости.

Использование проектов позволяет решить эти проблемы и имеет следующие преимущества:

- для администраторов платформы:
 - **единообразие** — администраторы могут создавать проекты, используя один и тот же шаблон, что обеспечивает единообразие и упрощает управление;
 - **безопасность** — проекты обеспечивают изоляцию ресурсов и политик доступа между различными проектами, что поддерживает безопасное многоотенантное окружение;
 - **потребление ресурсов** — администраторы могут легко устанавливать квоты на ресурсы и ограничения для каждого проекта, предотвращая избыточное использование ресурсов.
- для пользователей платформы:
 - **быстрый старт** — разработчики могут запрашивать у администраторов проекты, созданные по готовым шаблонам, что позволяет быстро начать разработку нового приложения;
 - **изоляция** — каждый проект обеспечивает изолированное окружение, где разработчики могут развертывать и тестировать свои приложения без влияния на другие проекты.



Копирование секретов во все неймспейсы не может использоваться совместно с проектами режима мультиотенантности.

В этом режиме создаются изолированные окружения для пользователей в их проектах, а `secret-copier` автоматически раздает секреты во все неймспейсы. Если в закрытом окружении пользователя окажутся чувствительные данные, это может привести к утечке данных и нарушению модели безопасности.

Ограничения

Проекты имеют ряд ограничений:

- Создание более одного неймспейса внутри проекта не предусматривается. Если требуется несколько неймспейсов, создайте отдельный проект для каждого из них.
- Ресурсы шаблона применяются только к одному неймспейсу, имя которого совпадает с именем проекта.

Дополнительные ресурсы

- Документация модуля `secret-copier`

Управление проектами

В Deckhouse Kubernetes Platform есть набор шаблонов для создания проектов:

- `default` — шаблон для базовых сценариев использования проектов:
 - ограничение ресурсов;
 - сетевая изоляция;
 - автоматические алерты и сбор логов;
 - выбор профиля безопасности;
 - настройка администраторов проекта.
- `secure` — включает все возможности шаблона `default`, а также дополнительные функции:
 - настройка допустимых для проекта UID/GID;
 - правила аудита обращения Linux-пользователей проекта к ядру;
 - сканирование запускаемых образов контейнеров на наличие известных уязвимостей (CVE).
- `secure-with-dedicated-nodes` — включает все возможности шаблона `secure`, а также дополнительные функции:
 - определение селектора узла для всех подов в проекте: если под создан, селектор узла пода будет автоматически **заменён** на селектор узла проекта;
 - определение стандартных tolerations для всех подов в проекте: если под создан, стандартные значения tolerations **добавляются** к нему автоматически.

Чтобы просмотреть все доступные параметры для шаблона проекта, выполните команду:

```
d8 k get projecttemplates <ИМЯ_ШАБЛОНА_ПРОЕКТА> -o
jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

Создание проекта

1. Для создания проекта создайте кастомный ресурс Project с указанием имени шаблона проекта в поле `.spec.projectTemplateName`.
2. В параметре `.spec.parameters` укажите значения параметров для секции `.spec.parametersSchema.openAPIV3Schema` кастомного ресурса ProjectTemplate.

Пример создания проекта с помощью Project из `default` ProjectTemplate представлен ниже:

```

apiVersion: deckhouse.io/v1alpha2
kind: Project
metadata:
  name: my-project
spec:
  description: This is an example from the Deckhouse documentation.
  projectTemplateName: default
  parameters:
    resourceQuota:
      requests:
        cpu: 5
        memory: 5Gi
        storage: 1Gi
      limits:
        cpu: 5
        memory: 5Gi
    networkPolicy: Isolated
    podSecurityProfile: Restricted
    extendedMonitoringEnabled: true
  administrators:
    - subject: Group
      name: k8s-admins

```

3. Для проверки статуса проекта выполните команду:

```
d8 k get projects my-project
```

Успешно созданный проект должен отображаться в статусе **Deployed** (синхронизирован). Если отображается статус **Error** (ошибка), добавьте аргумент `-o yaml` к команде (например, `d8 k get projects my-project -o yaml`) для получения более подробной информации о причине ошибки.

Автоматическое создание проекта для неймспейса

Для неймспейса возможно создать новый проект. Для этого пометьте неймспейс аннотацией `projects.deckhouse.io/adopt`. Например:

1. Создайте новый неймспейс:

```
d8 k create ns test
```

2. Пометьте его аннотацией:

```
d8 k annotate ns test projects.deckhouse.io/adopt=""
```

3. Убедитесь, что проект создан:

```
d8 k get projects
```

В списке проектов появится новый проект, соответствующий неймспейсу:

NAME	STATE	PROJECT TEMPLATE	DESCRIPTION	AGE
deckhouse-181d	Deployed	virtual	This is a virtual project	
default-181d	Deployed	virtual	This is a virtual project	
test-1m	Deployed	empty		

Шаблон созданного проекта можно изменить на существующий.



Обратите внимание, что при смене шаблона может возникнуть конфликт ресурсов: если в чарте шаблона прописаны ресурсы, которые уже присутствуют в неймспейсе, то применить шаблон не получится.

Создание собственного шаблона для проекта

Шаблоны проектов по умолчанию включают базовые сценарии использования и служат примером возможностей шаблонов.

Для создания своего шаблона:

1. Возьмите за основу один из шаблонов по умолчанию, например, `default`.
2. Скопируйте его в отдельный файл, например, `my-project-template.yaml` при помощи команды:

```
d8 k get projecttemplates default -O yaml > my-project-template.yaml
```

3. Отредактируйте файл `my-project-template.yaml`, внесите в него необходимые изменения.



Необходимо изменить не только шаблон, но и схему входных параметров под него. Шаблоны для проектов поддерживают все функции шаблонизации Helm.

4. Измените имя шаблона в поле `.metadata.name`.
5. Примените полученный шаблон командой:

```
d8 k apply -f my-project-template.yaml
```

6. Проверьте доступность нового шаблона с помощью команды:

```
d8 k get projecttemplates <ИМЯ_НОВОГО_ШАБЛОНА>
```

Использование лейблов для управления ресурсами

При создании ресурсов в ProjectTemplate можно использовать специальные лейблы для управления поведением `multitenancy-manager` при обработке этих ресурсов:

Пропуск создания лейбла `heritage: multitenancy-manager`

По умолчанию все ресурсы, созданные из ProjectTemplate, получают лейбл `heritage: multitenancy-manager`.

Он запрещает изменение ресурсов пользователями или любым контроллером, кроме `multitenancy-manager`.

Если необходимо разрешить изменение ресурса (например, для совместимости с другими системами, или в случае реализации собственного контроля изменения создаваемых объектов), добавьте к ресурсу лейбл `projects.deckhouse.io/skip-heritage-label`.

Пример:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
labels:
  projects.deckhouse.io/skip-heritage-label: "true"
  app: my-app
data:
  key: value
```

В этом случае ресурс получит лейблы `projects.deckhouse.io/project` и

`projects.deckhouse.io/project-template`, но не получит лейбл `heritage: multitenancy-manager`.

Исключение ресурсов из управления `multitenancy-manager`

Если необходимо исключить ресурс из управления `multitenancy-manager` (например, если он должен управляться вручную или другим контроллером), добавьте к ресурсу лейбл `projects.deckhouse.io/unmanaged`.

Пример:

```
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Ресурсы с лейблом `projects.deckhouse.io/unmanaged`:

- Будут созданы **только один раз** при создании проекта.
- **Не будут обновляться** при последующих изменениях шаблона или обновлениях.
- Не будут отслеживаться в статусе проекта.
- Получат лейблы `projects.deckhouse.io/project` и `projects.deckhouse.io/project-template`, но **не получают** лейбл `heritage: multitenancy-manager`.



После того как ресурс помечен как `unmanaged`, он будет создан при первой установке, но не будет обновляться при изменении `ProjectTemplate`. После создания ресурс становится полностью независимым и должен управляться вручную.

Реализация валидации изменений объектов с помощью пользовательского лейбла

Модуль `multitenancy-manager` использует `ValidatingAdmissionPolicy` для защиты ресурсов с лейблом `heritage: multitenancy-manager` от ручных изменений. Вы можете реализовать аналогичную валидацию для ресурсов с любым лейблом.

Принцип работы валидации в `multitenancy-manager`

`multitenancy-manager` валидирует объекты с лейблом `heritage: multitenancy-manager`. Для этого используются следующие ресурсы:

1. `ValidatingAdmissionPolicy` — определяет правила валидации:
 - Операции: `UPDATE` и `DELETE`;
 - Проверка: разрешены только операции от имени `service account` контроллера;
 - Применяется ко всем ресурсам и API группам.
2. `ValidatingAdmissionPolicyBinding` — определяет на какие объекты распространяется валидация:
 - Использует `namespaceSelector` и `objectSelector` для выбора ресурсов по лейблу `heritage: multitenancy-manager`.

Создание собственной валидации

Для реализации валидации для ресурсов с другим лейблом (например, `heritage: my-custom-label`):

1. Создайте файл с манифестами ресурсов `ValidatingAdmissionPolicy` и `ValidatingAdmissionPolicyBinding`:

```

apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:  ["*"]
        apiVersions: ["*"]
        operations: ["UPDATE", "DELETE"]
        resources:  ["*"]
        scope:      "*"
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-namespace:my-service-account"' # Замените на ваш сервисный аккаунт.
      reason: Forbidden
      messageExpression: 'object.kind == ''Namespace'' ? ''This resource is managed by '' + object.metadata.name + '' system. Manual modification is forbidden.'' : ''This resource is managed by '' + object.metadata.namespace + '' system. Manual modification is forbidden.''
    ---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    namespaceSelector:
      matchLabels:
        heritage: my-custom-label
    objectSelector:
      matchLabels:
        heritage: my-custom-label

```

2. Настройте параметры валидации:

- `policyName` — уникальное имя политики (должно совпадать с `Policy` и `Binding`);
- `request.userInfo.username` — имя сервисного аккаунта, которому разрешено изменять ресурсы (замените на ваш сервисный аккаунт);
- `heritage: my-custom-label` — значение лейбла `heritage` для ваших ресурсов (замените на ваше значение). Запрещено использовать значение `multitenancy-manager`, `deckhouse`;
- `failurePolicy: Fail` — политика при ошибке валидации:
 - `Fail` — отклонять запрос при ошибке проверки,
 - `Ignore` — игнорировать ошибки валидации.
- `validationActions` — действия валидации:
 - `Deny` — отклонять неразрешенные операции,
 - `Audit` — записывать операции в аудит лог.

3. Примените политику:

```
d8 k apply -f my-validation-policy.yaml
```

4. Убедитесь, что ваши ресурсы имеют соответствующий лейбл `heritage`:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label

```

Выдача кластерных ресурсов проектам

`multitenancy-manager` позволяет администраторам кластера управлять тем, какие кластерные ресурсы (например, `StorageClass`) можно использовать из неймспейсов проектов.

Для этого используются кастомные ресурсы:

- `GrantableClusterResourceDefinition (cluster-scoped)` — регистрирует кластерный ресурс, который можно выдавать проектам. Определяет тип ресурса (`grantedResource`), где проверяются ссылки на него (`usageReferences`), базовую доступность (`defaultAvailability`) и способ определения значения по умолчанию для проекта (`defaultFrom`). Каждая ссылка отдельно включает подстановку значения по умолчанию через `default: true`. Указывайте этот параметр только для полей, значение которых всегда должно быть задано (например, `storageClassName` у `PersistentVolumeClaim`). Если отсутствие ссылки имеет смысл (например, для аннотации-переключателя), не указывайте `default: true`. В этом случае такая ссылка по-прежнему будет проверяться и учитываться в квоте, но её значение не будет подставляться автоматически.
- `ClusterResourceGrantPolicy (cluster-scoped)` — выбирает проекты (по меткам неймспейса через `projectSelector`) и для каждого ресурса (`resourceName`) задаёт разрешённые имена (`allowed`, `allowedSelector`) и значение по умолчанию (`default`). Если задан `allow-лист`, использовать можно только ресурсы из этого списка.
- `AvailableClusterResource (namespaced, read-only, короткое имя available)` — формируемый контроллером каталог доступных для проекта кластерных ресурсов. Пользователи проекта могут использовать его, чтобы узнать, какие ресурсы им доступны.
- `ClusterResourceGrant (namespaced)` — определяет квоты проекта на кластерные ресурсы (лимиты на количество объектов и на измеряемые величины, например запрошенный объём хранилища). В статусе объекта отображается текущее потребление.

```

apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceDefinition
metadata:
  name: storageclasses
spec:
  grantedResource:
    apiGroup: storage.k8s.io
    kind: StorageClass
  enforcement: Managed
  defaultAvailability: All
  defaultFrom:
    annotationKey: storageclass.kubernetes.io/is-default-class
  usageReferences:
    - rule:
        apiGroups:
          - ""
        apiVersions:
          - v1
        resources:
          - persistentvolumeclaims
        fieldPath: $.spec.storageClassName
        default: true
    ---
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: ClusterResourceGrantPolicy
metadata:
  name: production-storage
spec:
  projectSelector:
    matchLabels:
      environment: production
  resources:
    - resourceName: storageclasses
      default: fast-ssd          # Перекрывает дефолт по аннотации.
      allowed:
        - fast-ssd
        - standard
      allowedSelector:          # Дополнительно разрешает любой StorageClass с меткой
shared=true.
        matchLabels:
          shared: "true"

```

Особенности применения:

- Проверяющий (validating) вебхук запрещает создание/обновление объектов в подходящих проектах, если используемое значение не разрешено. Уже присутствующие в объекте значения при обновлении не блокируются — существующие объекты продолжают работать.
- Мутирующий (mutating) вебхук подставляет значение по умолчанию только при создании и только в ссылки, помеченные `default: true`. Ссылки без неё (например, аннотации-переключатели) никогда не заполняются.
- Grant без совпавших проектов (или проект без совпавших grant'ов) ничего не ограничивает.

Дополнительные ресурсы

- Кастомные ресурсы модуля `multitenancy-manager`

Обзор

Managed-сервисы в Deckhouse Kubernetes Platform позволяют запускать в кластере готовые к использованию прикладные сервисы, например базы данных, очереди сообщений и сервисы обработки больших данных, без необходимости самостоятельно управлять их жизненным циклом. DKP берёт на себя технические задачи администрирования, масштабирования, резервного копирования и обновления, а администратор управляет managed-сервисами на

уровне классов, ограничений и архитектурных решений.

Как это работает

С помощью объектов кластера администратор настраивает managed-сервис в DKP, а пользователь описывает желаемое состояние managed-сервиса в неймспейсе. Специальный оператор отслеживает состояние объектов и управляет рабочими экземплярами managed-сервиса.

Используйте этот раздел, чтобы:

- подготовить кластер к работе с managed-сервисами;
- настроить общие параметры и ограничения managed-сервиса;
- определить доступные для использования конфигурации managed-сервисов;
- проверить готовность инфраструктуры к эксплуатации managed-сервисов.

Пользовательские инструкции по работе с managed-сервисами приведены в разделе «Использование».

Managed PostgreSQL

Сервис Managed PostgreSQL помогает разворачивать и поддерживать PostgreSQL в DKP. Поддерживаемая версия PostgreSQL — 17.6.

На этой странице описано, что и как может настраивать администратор кластера: управление ограничениями ресурсов и топологией, настройка проверки параметров, установка значений по умолчанию и привязка к узлам кластера.

Включение сервиса

Чтобы включить в DKP сервис Managed PostgreSQL, выполните следующие шаги:

1. Убедитесь, что выполнены требования к работе модуля `managed-postgres`.
2. Включите модуль `managed-postgres` в web-интерфейсе DKP или другими способами.
3. Создайте `PostgresClass` с необходимыми настройками, которые будут использоваться пользователями при создании объектов `Postgres`. Также можно использовать `PostgresClass default`, создаваемый модулем при включении. `PostgresClass default` содержит базовые настройки, чтобы пользователи могли сразу после включения Managed PostgreSQL в DKP создавать базы данных. Для production-окружений рекомендуется создавать отдельные `PostgresClass` с явными настройками и ограничениями.

После включения модуля пользователи смогут сами создавать базы данных PostgreSQL. Пользовательские операции с сервисом описаны в разделе «Использование» → «Managed-сервисы» → «Managed PostgreSQL».

Зависимости для отдельных функций

Некоторые функции модуля требуют дополнительной настройки компонентов Deckhouse Kubernetes Platform или инфраструктуры кластера:

Функция	Требование	
Резервное копирование (управляется <code>PostgresSnapshot</code>)	Включённый модуль <code>snapshot-controller</code> и <code>StorageClass</code> с поддержкой снимков	
TLS через <code>cert-manager</code>	Включённый модуль <code>cert-manager</code> и настроенный <code>ClusterIssuer</code> или <code>Issuer</code>	
Размещение на выделенных узлах	Лейблы на узлах (например, <code>node.deckhouse.io/group=pg</code>) и <code>taints</code> при необходимости. Подробнее — в документации	

Пример создания `PostgresClass`

Перед применением примера убедитесь, что в кластере достаточно ресурсов. Чтобы оценить доступные ресурсы, выполните:

```
d8 k describe node worker-0 | grep -A 5 "Allocated resources"
```

Пример вывода:

```
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests           Limits
-----
cpu                 3176m (80%)        500m (12%)
memory              8342837084 (71%)   6400Mi (57%)
```

Ниже приведён пример `PostgresClass production-v1`, который можно использовать вместо `PostgresClass default` и адаптировать под требования к ресурсам, топологии, параметрам PostgreSQL и размещению на узлах. Отдельные параметры примера подробнее описаны в следующих разделах.

Создайте файл `postgresclass-production.yaml` со следующим содержимым:

```
apiVersion: managed-services.deckhouse.io/v1alpha1
kind: PostgresClass
metadata:
  name: production-v1
spec:
  # Ограничение ресурсов CPU и памяти.
  sizingPolicies:
    - cores:
        min: 1
        max: 2
      memory:
        min: 512Mi
        max: 2Gi
        step: 512Mi
      coreFractions:
        - 50
        - 100
    - cores:
        min: 3
        max: 4
      memory:
        min: 2Gi
        max: 4Gi
        step: 1Gi
      coreFractions:
        - 50
        - 100

  # Управление отказоустойчивостью.
  topology:
    allowedTopologies:
      - Ignored
      - Zonal
      - TransZonal
    defaultTopology: TransZonal
    allowedZones:
      - zone-a
      - zone-b
      - zone-c

  # Проверка настроек PostgreSQL.
  validations:
    - message: "Max connections should not be more than 300"
      rule: "configuration.maxConnections <= 300"
    - message: "Shared buffers should not be more than 25% of RAM"
      rule: "configuration.sharedBuffers < instance.memory.size / 4"
    - message: "walKeepSize can not be more than 1Gi"
```

```

rule: "configuration.walKeepSize <= 1073741824"

# Параметры по умолчанию и разрешения на переопределение.
configuration:
  maxConnections: 200
  sharedBuffers: 1Gi
overridableConfiguration:
  - maxConnections
  - sharedBuffers
  - walKeepSize

# Привязка к выделенным узлам.
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
      - matchExpressions:
          - key: "node.deckhouse.io/group"
            operator: "In"
            values:
              - "pg"
              - "postgres"
    nodeSelector:
      "node.deckhouse.io/group": "pg"
tolerations:
  - key: primary-role
    operator: Equal
    value: pg
    effect: NoSchedule

```

Примените манифест:

```
d8 k apply -f postgresclass-production.yaml
```

Проверьте созданный PostgresClass:

```
d8 k describe postgresclass production-v1
```

Пример вывода...

```

Name:          production-v1
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   managed-services.deckhouse.io/v1alpha1
Kind:          PostgresClass
Metadata:
  Creation Timestamp:  2026-08-06T13:08:24Z
  Generation:         1
  Resource Version:    21610964
  UID:                f2548e11-a23c-4bd1-b786-776bf1677a1d
Сpec:
  Configuration:
    Max Connections:  200
    Shared Buffers:   1Gi
  Node Affinity:
    Required During Scheduling Ignored During Execution:
      Node Selector Terms:
        Match Expressions:
          Key:          node.deckhouse.io/group
          Operator:     In
          Values:
            pg

```



```

      postgres
Node Selector:
  node.deckhouse.io/group:  pg
Overridable Configuration:
  maxConnections
  sharedBuffers
  walKeepSize
Sizing Policies:
  Core Fractions:
    50
    100
  Cores:
    Max:  2
    Min:  1
  Memory:
    Max:   2Gi
    Min:  512Mi
    Step: 512Mi
  Core Fractions:
    50
    100
  Cores:
    Max:  4
    Min:  3
  Memory:
    Max:   4Gi
    Min:  2Gi
    Step: 1Gi
Tolerations:
  Effect:  NoSchedule
  Key:     primary-role
  Operator: Equal
  Value:   pg
Topology:
  Allowed Topologies:
    Ignored
    Zonal
    TransZonal
  Allowed Zones:
    zone-a
    zone-b
    zone-c
  Default Topology:  TransZonal
Validations:
  Message: Max connections should not be more than 300
  Rule:    configuration.maxConnections <= 300
  Message: Shared buffers should not be more than 25% of RAM
  Rule:    configuration.sharedBuffers < instance.memory.size / 4
  Message: walKeepSize can not be more than 1Gi
  Rule:    configuration.walKeepSize <= 1073741824
Events:    <none>

```

Пользователи могут использовать созданный PostgresClass, указывая его в параметре postgresClassName Postgres.



После создания PostgresClass его параметры нельзя изменить. Создайте новый PostgresClass с другим именем, чтобы использовать другие настройки.

Изменение и удаление PostgresClass

Чтобы удалить старый класс, сначала проверьте, используется ли он существующими объектами

Postgres. Выполните команду:

```
d8 k get postgres --all-namespaces -o custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,CLASS:.spec.postgresClassName | grep
production-v1
```

Если найдены объекты Postgres, использующие этот PostgresClass, учитывайте следующее:

- Базы данные, созданные на основе удалённого PostgresClass, продолжат работать. Их настройки остаются прежними, так как они были зафиксированы при создании.
- Пользователи не смогут создать новый объект Postgres со ссылкой на удалённый класс.

Если PostgresClass больше не используется его можно удалить.

Пример удаления PostgresClass `production-v1` через CLI:

```
d8 k delete postgresclass production-v1
```

Пример вывода:

```
postgresclass.managed-services.deckhouse.io "production-v1" deleted
```

i При необходимости изменения настроек PostgresClass рекомендуется создавать новый с другим именем, а не пересоздавать PostgresClass с прежним именем.

Ограничение ресурсов CPU и памяти

Для определения допустимых комбинаций CPU и памяти экземпляров PostgreSQL используются политики, в секции параметров `spec.sizingPolicies`. Администратор может задать несколько диапазонов ядер, для каждого из которых определены минимальная и максимальная память, а также шаг. Эти политики полезно применять, когда в одном кластере работают несколько независимых команд, создающих объекты Postgres без согласования с администратором. Политики предотвращают создание объектов Postgres с нереалистичными ресурсами и обеспечивают предсказуемую утилизацию узлов.

Выбор политики происходит по количеству ядер CPU. Параметр `coreFractions` определяет, какой процент от лимитов CPU (`limits`) составит гарантированный запрос (`requests`). Например, если администратор указал `coreFractions: [50, 100]`, пользователь при создании объекта Postgres может выбрать `coreFraction: 50` или `coreFraction: 100`.

При `coreFraction: 50`:

- `limits` CPU будут равны количеству запрошенных ядер (например, 4 ядра).
- `requests` CPU составят 50% от `limits` (2 ядра).

Планировщик гарантирует поду 2 ядра, но позволит использовать до 4, если они доступны. Это повышает плотность размещения подов на узлах.

При `coreFraction: 100`:

- `limits` и `requests` равны (4 ядра).
- Под получает гарантированное выделение ресурсов, но теряется возможность переиспользовать неиспользуемые ядра другими подами.

Ниже приведён фрагмент манифеста PostgresClass, с указанием допустимых комбинаций CPU и памяти. Полный вариант приведён в разделе «Пример создания PostgresClass».

```

spec:
  sizingPolicies:
    - cores:
        min: 1
        max: 2
      memory:
        min: 512Mi
        max: 2Gi
        step: 512Mi
      coreFractions:
        - 50
        - 100

```

В этом фрагменте для 1–2 ядер доступны значения памяти 512Mi, 1Gi, 1.5Gi и 2Gi.

Параметр `step` определяет шаг изменения объёма памяти. Пользователь может указать только то значение памяти, которое кратно `step`. Например, если `step: 512Mi`, допустимы значения 512Mi, 1Gi, 1.5Gi, 2Gi и так далее. Значение 700Mi будет отклонено, так как оно не кратно 512Mi.

Выбор памяти зависит от количества ядер. Если пользователь запросит 5 ядер, такой объект Postgres не будет создан, так как эта конфигурация не предусмотрена администратором.



В одном PostgresClass диапазоны `cores.min` – `cores.max` у разных политик не должны пересекаться.

Например, в PostgresClass `production-v1` первая политика не может покрывать 1–4 ядра, а вторая — 2–6 ядер, так как ядра 2–4 принадлежат обеим политикам. В разных PostgresClass, например `production-v1` и `production-v2`, диапазоны пересекаться могут.

Пользователь может увидеть доступные варианты и выбрать подходящий при создании объекта Postgres.

Управление отказоустойчивостью через зоны доступности

Администратор может управлять отказоустойчивостью экземпляров PostgreSQL пользователей, задавая топологию их распределения по зонам доступности в секции параметров `spec.topology`. Это может быть полезно в production-окружениях с требованиями к отказоустойчивости уровня дата-центра.

Доступны три топологии:

- `Ignored` — стандартное планирование без привязки к зонам;
- `Zonal` — все экземпляры размещаются в одной зоне (минимальная задержка между репликами). Подходит для сред с низкой задержкой, где потеря зоны допустима;
- `TransZonal` — экземпляры распределяются по разным зонам (один основной экземпляр, одна синхронная реплика, одна асинхронная реплика). Защищает от падения целой зоны, но требует больше ресурсов.

Администратор может указать разрешённые варианты топологий (`allowedTopologies`), топологию по умолчанию (`defaultTopology`) и список доступных зон (`allowedZones`).

Ниже приведён фрагмент манифеста PostgresClass, с указанием параметров топологии. Полный вариант приведён в разделе «Пример создания PostgresClass».

```

spec:
  topology:
    allowedTopologies:
      - Zonal
      - TransZonal
    defaultTopology: TransZonal
    allowedZones:
      - zone-a
      - zone-b

```

В этом фрагменте разрешены режимы `Zonal` и `TransZonal`, а по умолчанию применяется `TransZonal`. В поле `allowedZones` указаны абстрактные зоны — замените их на реальные

названия зон из используемого облачного провайдера или дата-центра.

Пользователь может выбрать уровень отказоустойчивости при создании объекта `Postgres`. Если не указать явно, применится значение по умолчанию (указанное в параметре `defaultTopology`).

Автоматическая проверка настроек PostgreSQL

Для запрещения использования пользователями нежелательные комбинации параметров PostgreSQL, администратор может задать правила валидации на языке Common Expression Language (далее — CEL) в секции параметров `spec.validations` `PostgresClass`. Эти правила проверяются в API-сервере до создания объектов `Postgres`.

Эти правила полезно применять для защиты от типичных ошибок конфигурации, которые могут привести к проблемам производительности. Например, слишком большое значение `shared_buffers` может оставить PostgreSQL недостаточно памяти для других операций, а большое количество подключений — увеличить общее потребление памяти.

В правилах доступны следующие переменные:

- `configuration.maxConnections`;
- `configuration.workMem`;
- `configuration.sharedBuffers`;
- `configuration.walKeepSize`;
- `instance.memory.size`;
- `instance.cpu.cores`.

Ниже приведён фрагмент манифеста `PostgresClass`, с указанием правил валидации. Полный набор правил приведён в разделе «Пример создания `PostgresClass`».

```
spec:
  validations:
    - message: "Shared buffers should not be more than 25% of RAM"
      rule: "configuration.sharedBuffers < instance.memory.size / 4"
```

В этом фрагменте правило запрещает выделять под `shared_buffers` более 25% от запрошенной памяти.

Пользователь получает ошибку валидации при попытке применить некорректные параметры, что помогает избежать ошибок конфигурации без ручной проверки.

Задание настроек по умолчанию и их переопределение

Для задания стандартных параметров PostgreSQL и определения, какие из них пользователь может менять, администратор может использовать параметры `spec.configuration` и `spec.overridableConfiguration`.

Если значения в `configuration` не указаны, контроллер модуля применяет следующие:

- `maxConnections`: 100;
- `sharedBuffers`: 25% от `memory.size`;
- `workMem`: $(\text{memory.size} - \text{sharedBuffers}) * 4 / \text{maxConnections}$;
- `walKeepSize`: 512Mi.

Ниже приведён фрагмент манифеста `PostgresClass`, с указанием настроек сервиса по умолчанию. Полный вариант приведён в разделе «Пример создания `PostgresClass`».

```
spec:
  configuration:
    maxConnections: 200
    sharedBuffers: 1Gi
  overridableConfiguration:
    - maxConnections
    - sharedBuffers
    - walKeepSize
```

В этом примере:

- `maxConnections` и `sharedBuffers` по умолчанию равны 200 и 1Gi соответственно.
- Пользователь может переопределить `maxConnections`, `sharedBuffers` и `walKeepSize`.
- `workMem` не входит в `overridableConfiguration` — пользователь не может его изменить.

Автоматический расчёт workMem

Для установки ограничения использования памяти операций сортировки и хеширования внутри одного запроса, используйте параметр `workMem`.

В PostgreSQL этот лимит применяется к каждой операции в каждой активной сессии. Сложный запрос может выполнять несколько таких операций одновременно. При большом количестве подключений суммарное потребление памяти может многократно превысить заданное значение и привести к исчерпанию памяти. Слишком маленькое значение, напротив, заставит PostgreSQL использовать временные файлы на диске и снизит производительность.

Если `workMem` не задан явно, модуль `managed-postgres` автоматически рассчитывает его значение на основе выделенных ресурсов.

Модуль рассчитывает `workMem` по следующей формуле:

```
workMem = (instance.memory.size - configuration.sharedBuffers) * 4 / configuration.maxConnections
```

Например, для Postgres со следующими параметрами:

- `instance.memory.size`: 4Gi
- `configuration.sharedBuffers`: 1Gi
- `configuration.maxConnections`: 200

Расчёт `workMem` будет выглядеть следующим образом:

```
workMem = (4Gi - 1Gi) * 4 / 200
workMem = 3Gi * 4 / 200
workMem = 12Gi / 200
workMem = 61.44Mi
```

Множитель `* 4` — коэффициент запаса на случай нескольких одновременных операций сортировки и хеширования в рамках одного запроса.

Контроллер рассчитывает `workMem` по приведённой выше формуле с учётом объёма памяти экземпляра, `sharedBuffers`, `maxConnections` и коэффициента запаса.

Привязка к выделенным узлам

Стандартные механизмы Kubernetes — `spec.nodeSelector`, `spec.tolerations` и `spec.nodeAffinity` — позволяют указать, на каких узлах могут размещаться поды PostgreSQL.

Размещение на выделенных узлах помогает изолировать экземпляры PostgreSQL от пользовательских приложений и сделать использование ресурсов дисковой подсистемы и сети более предсказуемым.

Ниже приведён фрагмент манифеста `PostgresClass`, с указанием `spec.nodeAffinity`, `spec.nodeSelector` и `spec.tolerations`. Полный вариант приведён в разделе «Пример создания `PostgresClass`».

```
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: "node.deckhouse.io/group"
              operator: "In"
              values:
                - "pg"
      nodeSelector:
        "node.deckhouse.io/group": "pg"
  tolerations:
    - key: primary-role
      operator: Equal
      value: pg
      effect: NoSchedule
```

В этом примере:

- Поды размещаются только на узлах с лейблом `node.deckhouse.io/group=pg`.
- На узлах установлен taint `primary-role=pg:NoSchedule`, который препятствует размещению подов без соответствующего `toleration`.
- `tolerations` разрешает размещение подов PostgreSQL на таких узлах.

Пользователь не заботится о выборе узлов. Поды автоматически размещаются на подготовленной инфраструктуре в соответствии с настройками `PostgresClass`.

Если правилам размещения не соответствует ни один узел, объекты `Postgres` пользователей останутся в состоянии `Pending`. Для поиска причины можно проверить события в неймспейсе, информацию об объекте `Postgres` и состояние `PVC`:

```
d8 k describe pods -n my-postgres <POD_NAME>
d8 k get events -n my-postgres --sort-by=.lastTimestamp
d8 k get pvc -n my-postgres
```

Дополнительные ресурсы

- Документация модуля `cert-manager`
- Документация модуля `managed-postgres`
- Документация модуля `snapshot-controller`

Резервное копирование и восстановление

Ручное восстановление кластера

Восстановление кластера с одним control-plane узлом

Для корректного восстановления кластера выполните следующие шаги на master-узле:

1. Подготовьте утилиту `etcdctl`. Найдите и скопируйте исполняемый файл на узле:

```
cp $(find /var/lib/containerd/ -name etcdctl -print -quit) /usr/local/bin/etcdctl
```

Проверьте версию `etcdctl`:

```
etcdctl version
```

Убедитесь, что результат команды `etcdctl version` отображается без ошибок.

При отсутствии `etcdctl` скачайте исполняемый файл из официального репозитория `etcd`, выбрав версию, которая соответствует версии `etcd` в кластере:

```
wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
tar -xzf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdctl /usr/local/bin/etcdctl
```

2. Проверьте версию `etcd` в кластере (при доступном Kubernetes API) выполнив команду:

```
d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdctl version
```

Если команда выполнится успешно, вы увидите актуальную версию `etcd`.

3. Остановите `etcd`. Переместите манифест `etcd`, чтобы kubelet прекратил запуск соответствующего пода с помощью команды:

```
mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
```

4. Убедитесь, что под `etcd` остановлен:

```
crictl ps | grep etcd
```

Если команда не возвращает данных о запущенном поде `etcd`, значит он успешно остановлен.

- Сохраните текущие данные etcd. Создайте резервную копию текущего состояния каталога `member`:

```
cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
```

В случае непредвиденных обстоятельств вы сможете вернуться к этим файлам.

- Переместите файл резервной копии etcd. Скопируйте или перенесите файл резервной копии etcd `etcd-backup.snapshot` в домашнюю директорию текущего пользователя (`root`):

```
cp /путь/до/резервной/копии/etcd-backup.snapshot ~/etcd-backup.snapshot
```

Убедитесь, что файл доступен для чтения:

```
ls -la ~/etcd-backup.snapshot
```

- Очистите директорию etcd. Удалите старые данные, чтобы подготовить etcd к восстановлению из резервной копии:

```
rm -rf /var/lib/etcd
```

Проверьте, что каталог `/var/lib/etcd` теперь пуст или отсутствует:

```
ls -la /var/lib/etcd
```

- Восстановите базу данных etcd из резервной копии. Воспользуйтесь `etcdctl` для восстановления:

```
ETCDCTL_API=3 etcdctl snapshot restore ~/etcd-backup.snapshot --data-dir=/var/lib/etcd
```

После завершения команды проверьте, что в каталоге `/var/lib/etcd/` появились файлы, соответствующие восстановленному состоянию.

- Запустите etcd. Верните манифест etcd в рабочую директорию, чтобы kubelet вновь запустил под etcd:

```
mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
```

- Дождитесь, пока под будет создан и перейдёт в состояние `Running`. Убедитесь, что он действительно запущен:

```
crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
```

Процесс запуска может занять некоторое время. После успешного старта etcd кластер будет восстановлен из резервной копии.

Пример вывода:

CONTAINER	IMAGE	CREATED	STATE	NAME
ATTEMPT	POD ID	POD		
4b11d6ea0338f	16d0a07aa1e26	About a minute ago	Running	etcd
ee3c8c7d7bba6	etcd-gs-test			0

- Перезапустите master-узел.

Восстановление мультимастерного кластера

Для корректного восстановления мультимастерного кластера выполните следующие шаги:

- Активируйте режим High Availability (HA). Это необходимо, чтобы сохранить хотя бы одну реплику Prometheus и его PVC, поскольку в кластере с одним master-узлом HA по умолчанию отключён.
- Переведите кластер в режим с одним master-узлом:

- В облачном кластере воспользуйтесь инструкцией.
 - В статическом кластере выведите лишние master-узлы из роли `control-plane` по инструкции, после чего удалите их из кластера.
 - В статическом кластере с настроенным режимом HA на базе двух master-узлов и arbiter-узла удалите arbiter-узел и лишние master-узлы.
 - В облачном кластере с настроенным режимом HA на базе двух master-узлов и arbiter-узла воспользуйтесь инструкцией для удаления лишних мастер-узлов и arbiter-узла.
3. Восстановите etcd из резервной копии на единственном оставшемся master-узле. Следуйте инструкции для кластера с одним control-plane узлом.
 4. Когда работа etcd будет восстановлена, удалите из кластера информацию об уже удаленных в первом пункте master-узлах, воспользовавшись следующей командой (укажите название узла):

```
d8 k delete node <ИМЯ_MASTER_УЗЛА>
```

i **Внимание.** При недоступности команд `d8 k` или `kubectl` на узле проверьте файл конфигурации `/etc/kubernetes/kubernetes-api-proxy/upstreams.json`. В нем должен быть указан только ваш текущий API-сервер. Если в конфигурации содержатся строки с IP-адресами старых master-узлов, удалите их. Аналогичным образом исправьте конфигурацию на всех остальных узлах.

5. Перезапустите master-узел. Убедитесь, что остальные узлы перешли в состояние `Ready`.
6. Дождитесь выполнения заданий из очереди Deckhouse:

```
d8 system queue main
```

7. Переведите кластер обратно в мультимастерный режим. Для облачных кластеров используйте инструкцию.

После этих шагов кластер будет успешно восстановлен в мультимастерной конфигурации.

Восстановление отдельных объектов

Восстановление объектов Kubernetes из резервной копии etcd

Чтобы восстановить отдельные объекты кластера (например, конкретные Deployment, Secret или ConfigMap) из резервной копии etcd, выполните следующие шаги:

1. Запустите временный экземпляр etcd. Создайте отдельную копию etcd, которая будет работать независимо от основного кластера.
2. Загрузите во временный экземпляр etcd данные из резервной копии. Используйте существующий файл резервной копии etcd, чтобы заполнить временный экземпляр нужными данными.
3. Выгрузите манифесты необходимых объектов в формате YAML.
4. Восстановите объекты кластера из выгруженных YAML-файлов.

Пример шагов по восстановлению объектов из резервной копии etcd

В следующем примере `etcd-backup.snapshot` — файл резервной копии etcd, `infra-production` — неймспейс, в котором нужно восстановить объекты.

- Для выгрузки бинарных данных из etcd потребуется утилита `auger`. Соберите ее из исходного кода на любой машине с Docker (на узлах кластера это сделать невозможно) с помощью следующих команд:

```
git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
cd auger
make release
build/auger -h
```

- Получившийся исполняемый файл `build/auger`, а также файл резервной копии etcd загрузите на master-узел, с которого будут выполняться дальнейшие действия.

Действия ниже выполняются на master-узле в кластере, на который предварительно был загружен файл резервной копии etcd и утилита `auger`:

1. Установите корректные права доступа для файла резервной копии etcd:

```
chmod 644 etcd-backup.snapshot
```

2. Установите полный путь до файла резервной копии etcd и до утилиты в переменных окружения:

```
SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot  
AUGER_BIN=/root/auger  
chmod +x $AUGER_BIN
```

3. Запустите под с временным экземпляром etcd:

- Создайте манифест пода. Он будет запускаться именно на текущем master-узле, выбрав его по переменной `$HOSTNAME`, и смонтирует файл резервной копии etcd по пути `$SNAPSHOT` для загрузки во временный экземпляр etcd:

```

cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
  - operator: Exists
  initContainers:
  - command:
    - etcdctl
    - snapshot
    - restore
    - "/tmp/etcd-snapshot"
    - "--data-dir=/default.etcd"
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
    {.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-snapshot-restore
    # Раскомментируйте фрагмент ниже, чтобы задать лимиты для контейнера,
    # если ресурсов узла недостаточно для его запуска.
    # resources:
    #   requests:
    #     ephemeral-storage: "200Mi"
    #   limits:
    #     ephemeral-storage: "500Mi"
    volumeMounts:
    - name: etcd-dir
      mountPath: /default.etcd
    - name: etcd-snapshot
      mountPath: /tmp/etcd-snapshot
      readOnly: true
  containers:
  - command:
    - etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
    {.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-temp
    volumeMounts:
    - name: etcd-dir
      mountPath: /default.etcd
  volumes:
  - name: etcd-dir
    emptyDir: {}
    # Используйте фрагмент ниже вместо emptyDir: {}, чтобы задать лимиты
    # для контейнера, если ресурсов узла недостаточно для его запуска.
    # emptyDir:
    #   sizeLimit: 500Mi
  - name: etcd-snapshot
    hostPath:
      path: $SNAPSHOT
      type: File
EOF

```

- Запустите под:

```
d8 k create -f etcd.pod.yaml
```

4. Установите нужные переменные. В текущем примере:

- `infra-production` — неймспейс, в пределах которого выполняется поиск ресурсов.
- `/root/etcd-restore/output` — каталог для восстановленных манифестов.
- `/root/auger` — путь до исполняемого файла утилиты `auger`:

```
FILTER=infra-production
BACKUP_OUTPUT_DIR=/root/etcd-restore/output
mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
```

5. Выполните команды, которые отфильтруют список нужных ресурсов по переменной `$FILTER` и выгрузят их в каталог `$BACKUP_OUTPUT_DIR`:

```
files=$(kubectl -n default exec etcdrestore -C etcd-temp -- etcdctl --
endpoints=localhost:2379 get / --prefix --keys-only | grep "$FILTER")
for file in "${files[@]}"
do
    OBJECT=$(kubectl -n default exec etcdrestore -C etcd-temp -- etcdctl --
endpoints=localhost:2379 get "$file" --print-value-only | $AUGER_BIN decode)
    FILENAME=$(echo $file | sed -e "s#/registry/##g;s#/#_#g")
    echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
    echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
done
```

6. Восстановите объекты кластера из выгруженных YAML-файлов.

7. Удалите под с временным экземпляром etcd:

```
d8 k -n default delete pod etcdrestore
```

Восстановление объектов кластера из выгруженных YAML-файлов

Для восстановления объектов выполните следующие шаги:

1. Подготовьте YAML-файлы к восстановлению. Перед тем как загружать объекты обратно в кластер, удалите из их описаний технические поля, которые могли устареть или нарушить процесс восстановления:

- `creationTimestamp`;
- `UID`;
- `status`.

Редактирование можно выполнить вручную или с помощью утилит для обработки YAML/JSON (например, `yq` или `jq`).

2. Создайте объекты в кластере. Для восстановления ресурсов выполните команду:

```
d8 k create -f <ПУТЬ_К_ФАЙЛУ>.yaml
```

При необходимости можно указать путь к конкретному файлу или каталогу.

3. Если нужно массово восстановить сразу несколько объектов, воспользуйтесь утилитой `find`:

```
find $BACKUP_OUTPUT_DIR -type f -name "*.yaml" -exec d8 k create -f {} \;
```

Эта команда найдёт все `.yaml`-файлы в заданном каталоге `$BACKUP_OUTPUT_DIR` и поочерёдно применит к ним `d8 k create`.

После выполнения этих шагов выбранные объекты будут воссозданы в кластере согласно описаниям из YAML-файлов.

Восстановление объектов при смене IP-адреса master-узла



Этот раздел описывает ситуацию, когда меняется только IP-адрес master-узла, а все остальные объекты в резервной копии etcd (например, CA-сертификаты) остаются валидными. Предполагается, что восстановление выполняется в кластере с одним master-узлом.

Для восстановления объектов из резервной копии etcd при смене IP-адреса выполните шаги:

1. Восстановите etcd из резервной копии. Следуйте стандартной процедуре восстановления etcd с использованием файла резервной копии etcd. Убедитесь, что на этапе восстановления вы не изменяете никаких других параметров, кроме данных etcd.
2. Обновите IP-адрес в статичных конфигурационных файлах:
 - Проверьте файлы манифестов компонентов Kubernetes, расположенные в `/etc/kubernetes/manifests/`.
 - Проверьте системные настройки kubelet (файлы в `/etc/systemd/system/kubelet.service.d/` или аналогичные директории).
 - При необходимости измените IP-адрес и в других конфигурациях, если они ссылаются на старый адрес.
3. Перевыпустите сертификаты, выданные для старого IP. Удалите или переместите старые сертификаты, связанные с `apiserver` и, при необходимости, с etcd. Сгенерируйте новые сертификаты, указав в качестве SAN (Subject Alternative Name) новый IP-адрес master-узла.
4. Перезапустите все сервисы, использующие обновлённые конфигурации и сертификаты. Заставьте kubelet перезапустить манифесты control-plane (API-сервер, etcd и т.д.). Перезагрузите системные службы (например, `systemctl restart kubelet`) или убедитесь, что все нужные процессы перезапущены автоматически.
5. Дождитесь, пока kubelet обновит собственный сертификат.

Данные действия можно произвести как автоматизировано — с помощью скрипта, так и вручную — путем выполнения одиночных команд.

Автоматизированная выгрузка объектов при смене IP-адреса

Чтобы упростить процесс восстановления кластера при смене IP-адреса master-узла, воспользуйтесь готовым скриптом, представленным ниже. Перед запуском скрипта:

1. Укажите корректные пути и IP-адреса:
 - `ETCD_SNAPSHOT_PATH` — путь до файла резервной копии etcd.
 - `OLD_IP` — старый IP-адрес master-узла, под которым создавалась резервная копия.
 - `NEW_IP` — новый IP-адрес master-узла.
2. Убедитесь, что версия Kubernetes (`KUBERNETES_VERSION`) совпадает с установленной в кластере. Это необходимо для корректной загрузки соответствующей версии kubeadm.
3. Скачайте утилиту `etcdctl`, если она не установлена.
4. После выполнения скрипта необходимо дождаться, пока kubelet обновит свой сертификат, учитывающий новый IP-адрес. Проверить это можно в директории `/var/lib/kubelet/pki/`, где должен появиться новый сертификат.

Скрипт для выгрузки объектов

```

ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Путь до файла резервной копии etcd.
OLD_IP=10.242.32.34                        # IP-адрес старого master-узла.
NEW_IP=10.242.32.21                        # IP-адрес нового master-узла.
KUBERNETES_VERSION=1.28.0                  # Версия Kubernetes.

mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
mkdir ./etcd_old
mv /var/lib/etcd ~/etcd_old
ETCDUTL_PATH=$(find /var/lib/containerd/ -name etcdctl -print -quit)

ETCDCTL_API=3 $ETCDUTL_PATH snapshot restore etcd-backup.snapshot --data-dir=/var/lib/etcd

mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'

mkdir -p ./old_certs/etcd
mv /etc/kubernetes/pki/apiserver.* ./old_certs/
mv /etc/kubernetes/pki/etcd/server.* ./old_certs/etcd/
mv /etc/kubernetes/pki/etcd/peer.* ./old_certs/etcd/

curl -LO https://dl.k8s.io/v$KUBERNETES_VERSION/bin/linux/amd64/kubeadm
chmod +x kubeadm
./kubeadm init phase certs all --config /etc/kubernetes/deckhouse/kubeadm/config.yaml

crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

systemctl daemon-reload
systemctl restart kubelet.service

```

Ручное восстановление объектов при смене IP-адреса

Если вы хотите вручную внести изменения при восстановлении кластера с обновлённым IP-адресом master-узла, выполните следующие действия:

1. Восстановите etcd из резервной копии:

- Переместите манифест etcd так, чтобы kubelet остановил соответствующий под:

```
mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
```

- Создайте директорию для резервного хранения прежних данных:

```
mkdir ./etcd_old
mv /var/lib/etcd ./etcd_old
```

- Найдите или скачайте утилиту `etcdctl`, если она не установлена, и выполните восстановление из снапшота:

```

ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Путь до файла резервной копии etcd.
ETCDUTL_PATH=$(find /var/lib/containerd/ -name etcdctl -print -quit)

ETCDCTL_API=3 $ETCDUTL_PATH snapshot restore \
    etcd-backup.snapshot \
    --data-dir=/var/lib/etcd

```

- Верните манифест etcd на место, чтобы kubelet снова запустил под:

```
mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
```

- Убедитесь, что etcd успешно запустился, проверив список подов с помощью `crictl ps | grep etcd` или просмотрев логи kubelet.
2. Обновите IP-адреса в статичных конфигурационных файлах. Если в манифестах или системных сервисах kubelet прописан старый IP-адрес, замените его на новый:

```
OLD_IP=10.242.32.34 # Старый IP-адрес master-узла.
NEW_IP=10.242.32.21 # Новый IP-адрес master-узла.

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'

```

3. Перевыпустите сертификаты, выпущенные для старого IP-адреса:

- Подготовьте каталог для временного хранения старых сертификатов:

```
mkdir -p ./old_certs/etcd
mv /etc/kubernetes/pki/apiserver.* ./old_certs/
mv /etc/kubernetes/pki/etcd/server.* ./old_certs/etcd/
mv /etc/kubernetes/pki/etcd/peer.* ./old_certs/etcd/

```

- Установите или скачайте kubeadm в соответствии с текущей версией Kubernetes:

```
KUBERNETES_VERSION=1.28.0 # Версия Kubernetes.
curl -LO https://dl.k8s.io/v$KUBERNETES_VERSION/bin/linux/amd64/kubeadm
chmod +x kubeadm

```

- Сгенерируйте новые сертификаты:

```
./kubeadm init phase certs all --config /etc/kubernetes/deckhouse/kubeadm/config.yaml
```

В созданных сертификатах будет учтён новый IP-адрес.

4. Перезапустите сервисы, использующие обновлённые конфигурации и сертификаты. Для немедленного прекращения работы запущенных контейнеров выполните:

```
crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

systemctl daemon-reload
systemctl restart kubelet.service

```

Kubelet перезапустит соответствующие поды, а компоненты Kubernetes загрузят новые сертификаты.

5. Дождитесь, пока kubelet обновит собственный сертификат. Kubelet автоматически генерирует и обновляет свой сертификат, в котором будет прописан новый IP-адрес:

- Проверьте директорию `/var/lib/kubelet/pki/`.
- Убедитесь, что новый сертификат присутствует и валиден.

После выполнения всех вышеперечисленных шагов кластер будет успешно восстановлен и продолжит работу с новым IP-адресом master-узла.

Создание резервных копий с помощью Deckhouse CLI

Deckhouse CLI (d8) предоставляет команду `backup` для создания резервных копий различных компонентов кластера:

- `etcd` — снимок ключевого хранилища данных Deckhouse;
- `cluster-config` — архив с ключевыми конфигурационными объектами кластера;
- `loki` — выгрузка логов из встроенного API Loki.

Резервное копирование etcd

Снимок etcd позволяет сохранить текущее состояние кластера на уровне key-value хранилища. Это полный дамп, который можно использовать для восстановления.

Для создания резервной копии выполните команду:

```
d8 backup etcd <путь-до-файла-резервной-копии-etcd> [флаги]
```

Флаги:

- `-p, --etcd-pod string` — имя пода etcd, из которого необходимо создать файл резервной копии etcd;
- `-h, --help` — показать справку по команде etcd;
- `--verbose` — подробный (расширенный) вывод логов.

Пример:

```
d8 backup etcd etcd-backup.snapshot
```

Пример вывода команды:

```
2025/04/22 08:38:58 Trying to snapshot etcd-sandbox-master-0
2025/04/22 08:39:01 Snapshot successfully taken from etcd-sandbox-master-0
```

Автоматическое резервное копирование etcd

Deckhouse автоматически выполняет ежедневное резервное копирование etcd с помощью CronJob, запускаемого в поде `d8-etcd-backup` в неймспейсе `kube-system`. В рамках задания создаётся снимок базы данных, архивируется и сохраняется локально на узле в директории `/var/lib/etcd/`:

```
etcdctl snapshot save etcd-backup.snapshot
tar -czvf etcd-backup.tar.gz etcd-backup.snapshot
mv etcd-backup.tar.gz /var/lib/etcd/etcd-backup.tar.gz
```

Для настройки автоматического резервного копирования используется модуль `control-plane-manager`. Необходимые параметры задаются в его конфигурации:

Параметр	Описание
<code>etcd.backup.enabled</code>	Включает ежедневное резервное копирование etcd.
<code>etcd.backup.cronSchedule</code>	Расписание выполнения резервного копирования в формате cron. Используется локальное время <code>kube-controller-manager</code> .
<code>etcd.backup.hostPath</code>	Путь на мастер-узлах, где будут сохраняться архивы резервных копий etcd.

Пример фрагмента конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    etcd:
      backup:
        cronSchedule: 0 14 * * *
        enabled: true
```

Резервное копирование конфигурации кластера

Команда `d8 backup cluster-config` создаёт архив с набором ключевых ресурсов, относящихся к конфигурации кластера. Это не полная резервная копия всех объектов, а определённый whitelist.

Для создания резервной копии выполните команду:

```
d8 backup cluster-config <путь-до-резервной-копии>
```

Пример:

```
d8 backup cluster-config /backup/cluster-config-2025-04-21.tar
```

В архив включаются только те объекты, которые соответствуют следующим критериям:

- Объекты CustomResource, чьи CRD помечены аннотацией:

```
backup.deckhouse.io/cluster-config=true
```

- StorageClass'ы, имеющие лейбл:

```
heritage=deckhouse
```

- Секреты и ConfigMap'ы, из неймспейсов, начинающихся на `d8-` или `kube-`, если они явно перечислены в файле `whitelist`.
- Роли и биндинги уровня кластера (ClusterRole и ClusterRoleBinding), если они не помечены лейблом:

```
heritage=deckhouse
```



Резервная копия включает только объекты CR, но не сами определения CRD. Для полного восстановления кластера CRD должны быть заранее установлены (например, из манифестов модулей Deckhouse).

Пример содержимого `whitelist`:

Неймспейс	Объект	Название
d8-system	Secret	d8-cluster-terraform-state
		<code>\$regexp:d8-node-terraform-state-(.*)\$</code>
		deckhouse-registry
	ConfigMap	d8-deckhouse-version-info
kube-system	ConfigMap	d8-cluster-is-bootstrapped

Неймспейс	Объект	Название
		d8-cluster-uuid
		extension-apiserver-authentication
	Secret	d8-cloud-provider-discovery-data
		d8-cluster-configuration
		d8-cni-configuration
		d8-control-plane-manager-config
		d8-node-manager-cloud-provider
		d8-pki
		d8-provider-cluster-configuration
		d8-static-cluster-configuration
		d8-secret-encryption-key
d8-cert-manager	Secret	cert-manager-letsencrypt-private-key
		selfsigned-ca-key-pair

Выгрузка логов из Loki

Команда `d8 backup loki` предназначена для выгрузки логов из встроенного Loki. Это не полноценная резервная копия, а лишь диагностическая выгрузка: полученные данные нельзя восстановить обратно в Loki.

Для успешной выгрузки `d8` обращается к Loki API от имени ServiceAccount `loki` в неймспейсе `d8-monitoring`, используя секрет с токеном.

ServiceAccount `loki` создаётся автоматически с версии Deckhouse v1.69.0. Однако для работы команды `d8 backup loki` необходимо вручную создать секрет и назначить Role и RoleBinding, если они ещё не заданы.

Примените манифесты перед запуском `d8 backup loki`, чтобы команда корректно получала токен и могла обращаться к Loki API.

Пример манифестов:

```

---
apiVersion: v1
kind: Secret
metadata:
  name: loki-api-token
  namespace: d8-monitoring
  annotations:
    kubernetes.io/service-account.name: loki
type: kubernetes.io/service-account-token
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
rules:
  - apiGroups: ["apps"]
    resources:
      - "statefulsets/http"
    resourceNames: ["loki"]
    verbs: ["create", "get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: access-to-loki-from-d8
subjects:
  - kind: ServiceAccount
    name: loki
    namespace: d8-monitoring

```

Для создания резервной копии выполните команду:

```
d8 backup loki [флаги]
```

Пример:

```
d8 backup loki --days 1 > ./loki.log
```

Флаги:

- `--start`, `--end` — временные лейблы в формате “YYYY-MM-DD HH:MM:SS”;
- `--days` — ширина временного окна выгрузки (по умолчанию 5 дней);
- `--limit` — максимум строк в одном запросе (по умолчанию 5000).

Список доступных флагов можно получить через следующую команду:

```
d8 backup loki --help
```

Дополнительные ресурсы

- Документация модуля `control-plane-manager`

Автоматическое назначение лейблов и аннотаций пространствам имён

Вы можете автоматизировать назначение лейблов и аннотаций пространствам имён в кластере Deckhouse на основе заданных шаблонов. Это полезно, например, когда нужно автоматически включать новые пространства имён в мониторинг без ручного редактирования каждого из них.

Как это работает

- Все пространства имён, чьи названия совпадают с шаблонами в `includeNames` и не совпадают с шаблонами в `excludeNames`, получают указанные лейблы и аннотации.
- При изменении конфигурации лейблы и аннотации на существующих пространствах имён обновляются автоматически.
- Новые пространства имён, подходящие под условия шаблонов, также получают нужные лейблы и аннотации автоматически.

Настройка автоматического назначения лейблов и аннотаций

Включите модуль `namespace-configurator`:

```
d8 system module enable namespace-configurator
```

Настройте автоматическое назначение лейблов и аннотаций в ModuleConfig `namespace-configurator`:

1. Перечислите аннотации и лейблы, которые должны применяться к пространствам имён, в полях `settings.configurations.annotations` и `settings.configurations.labels` соответственно;
2. Укажите шаблоны названий пространств имён, к которым должны применяться настройки:
 - в поле `includeNames` перечислите регулярные выражения, соответствующие нужным названиям;
 - в поле `excludeNames` перечислите те, которые нужно исключить.

Пример конфигурации

В следующем примере конфигурации настраивается автоматическое добавление лейбла `extended-monitoring.deckhouse.io/enabled=true` и аннотации `foo=bar` ко всем пространствам имён, названия которых начинаются с `prod-` или `infra-`, за исключением `infra-test`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
      - annotations:
          foo: bar
        labels:
          extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
        - "^prod"
        - "^infra"
      excludeNames:
        - "infra-test"
```

Дополнительные ресурсы

- Документация модуля `namespace-configurator`

Доставка приложений с помощью Argo CD

В этом разделе описана доставка приложений с помощью Argo CD в Deckhouse Kubernetes Platform (DKP).

Argo CD — это Open Source-инструмент для непрерывной доставки приложений в Kubernetes, реализующий подход GitOps. Git-репозиторий выступает источником истины для описания

приложений, их конфигурации и целевых окружений.

Argo CD отслеживает изменения, показывает расхождения между желаемым и фактическим состоянием ресурсов и при необходимости автоматически или вручную приводит кластер к целевому состоянию.

Основные возможности Argo CD:

- автоматическое и ручное развёртывание приложений в Kubernetes;
- визуализация различий между желаемым и фактическим состоянием;
- отслеживание конфигурации приложений в Git;
- работа с несколькими кластерами Kubernetes;
- управление доступом через собственную ролевую модель;
- откат к ранее зафиксированной конфигурации и аудит действий;
- работа через веб-интерфейс, CLI и API, включая интеграцию с Git-вебхуками и внешними системами автоматизации.

В DKP Argo CD запускается с помощью модуля `operator-argo`, основанного на проекте Argo CD Operator. Модуль позволяет декларативно развернуть и обслуживать один или несколько экземпляров Argo CD в кластере DKP без ручной установки и сопровождения компонентов.

Дополнительно модуль обеспечивает:

- поддержку единого входа (SSO);
- интеграцию с мониторингом в составе DKP;
- режим высокой доступности;
- поддержку автоматического обновления образов через Argo CD Image Updater;
- контроль доступа к кластерным ресурсам.

Типичный сценарий настройки со стороны администратора:

1. Включите модуль `operator-argo` и создайте объект ArgoCD.
2. Настройте аутентификацию пользователей.
3. Настройте ролевую модель доступа.
4. Передайте пользователям доступ к экземпляру Argo CD для доставки приложений (подробнее — в разделе «Использование»).

Устройство модуля `operator-argo` описано в разделе «Архитектура». Параметры модуля и кастомных ресурсов приведены в документации модуля `operator-argo`.

Дополнительные ресурсы

- Официальный сайт Argo CD
- Официальный сайт Argo CD Operator
- Документация модуля `operator-argo`

Запуск Argo CD

В этом разделе описаны шаги, необходимые для запуска экземпляра Argo CD в кластере DKP:

- Подготовка к запуску Argo CD;
- Развёртывание одного или нескольких экземпляров Argo CD;
- Дополнительные настройки Argo CD.

Подготовка к запуску Argo CD

Прежде чем создавать экземпляры Argo CD, выполните следующие шаги:

1. Включите модуль `operator-argo`.
2. Дождитесь, пока модуль перейдёт в состояние `Ready`.

Проверить состояние модуля можно в веб-интерфейсе DKP или с помощью команды:

```
d8 k get module operator-argo -w
```

Подробная информация о настройках модуля приведена в документации модуля `operator-argo`.

После включения модуля `operator-argo` в кластере DKP станут доступны кастомные ресурсы Argo CD.

Чтобы запустить экземпляр Argo CD, необходимо создать объект ArgoCD. Работа с кастомными ресурсами, относящимися к области экземпляра Argo CD, описана в разделе «Использование».

Список кастомных ресурсов Argo CD, создаваемых модулем `operator-argo`...

CRD	Область использования	Назначение
<code>argocds.argoproj.io</code>	Модуль operator-argo	Кастомный ресурс, по которому модуль <code>operator-argo</code> настраивает и сопровождает экземпляр Argo CD в целевой среде. Через объект ArgoCD задаются параметры установки, состав компонентов, интеграции, политики доступа и другие настройки экземпляра.
<code>appprojects.argoproj.io</code>	Экземпляр Argo CD	Используется для логической сегментации приложений и задания политик: допустимых Git-репозиторий, целевых кластеров и неймспейсов, а также правил доступа и ограничений на использование ресурсов.
<code>applications.argoproj.io</code>	Экземпляр Argo CD	Основная прикладная CRD Argo CD для описания приложения, которое должно синхронизироваться из декларативного источника (Git, Helm, Kustomize и другие) в Kubernetes. Определяет источник, целевое окружение и параметры синхронизации.
<code>applicationsets.argoproj.io</code>	Экземпляр Argo CD	Используется для автоматизированного создания набора объектов Application по шаблону. Подходит для сценариев массового управления приложениями: в нескольких кластерах, окружениях, директориях, командах или ветках репозитория.
<code>argocdexports.argoproj.io</code>	Экземпляр Argo CD	Используется для декларативного экспорта данных, связанных с экземпляром Argo CD, во внешние системы или смежные компоненты DKP. Обычно применяется в интеграционных сценариях, где требуется формализованно публиковать сведения о конфигурации, статусе или доступах.
<code>namespacemanagements.argoproj.io</code>	Экземпляр Argo CD	Используется для управления жизненным циклом неймспейсов в рамках GitOps-процессов. Может автоматизировать создание, настройку и сопровождение неймспейсов, в которые затем выполняется развёртывание приложений.

CRD	Область использования	Назначение
<code>notificationsconfiguration.s.argoproj.io</code>	Экземпляр Argo CD	Используется для настройки механизма уведомлений Argo CD. Позволяет декларативно описывать каналы доставки и правила отправки событий, связанных с синхронизацией, ошибками развёртывания, изменением статуса приложений и другими операционными событиями.
<code>imageupdaters.argocd-image-updater.s.argoproj.io</code>	Экземпляр Argo CD	Используется для автоматического отслеживания новых версий контейнерных образов и обновления параметров приложений в соответствии с заданной стратегией версионирования и публикации.

Развёртывание экземпляра Argo CD

Параметры, доступные для конфигурации экземпляра Argo CD, приведены в документации модуля `operator-argo`.

Чтобы развернуть экземпляр Argo CD в неймспейсе `argocd` и опубликовать веб-интерфейс Argo CD через Ingress, используйте следующий пример (укажите собственные значения для `<ARGOCD_DOMAIN>` и `<TLS_SECRET_NAME>`):

```

apiVersion: v1
kind: Namespace
metadata:
  name: argocd
---
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: <TLS_SECRET_NAME>
    insecure: true
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: argocd-ingress
  namespace: argocd
spec:
  dnsNames:
    - <ARGOCD_DOMAIN>
  issuerRef:
    kind: ClusterIssuer
    name: letsencrypt
  secretName: <TLS_SECRET_NAME>

```



Параметр `spec.server.insecure: true` в примере выше отключает внутренний TLS у API-сервера Argo CD. Это позволяет избежать циклических перенаправлений при публикации через Ingress.

После создания объекта ArgoCD в неймспейсе `argocd` будут запущены компоненты Argo CD:

```

d8 k -n argocd get pods

```

NAME	READY	STATUS	RESTARTS	AGE
argocd-application-controller-0	1/1	Running	0	35m
argocd-dex-server-759fff8444-zglp4	1/1	Running	0	2d23h
argocd-redis-568f5b889c-jg5dr	1/1	Running	0	4d
argocd-repo-server-78d9d6bcc6-9rwcw	1/1	Running	0	3d22h
argocd-server-76597597f9-kfqdl	1/1	Running	0	35m
podinfo-ccdb96645-zv5tm	1/1	Running	0	2d20h

Когда все компоненты перейдут в статус `Running`, веб-интерфейс экземпляра Argo CD станет доступен по адресу, указанному в параметре `spec.server.ingress.tls.hosts` (в примере — `https://<ARGOCD_DOMAIN>`). Настройка аутентификации и получения учётных данных описана в разделе «Настройка аутентификации и авторизации».

Доставка пользовательских приложений с помощью Argo CD описана в разделе «Использование».

i Для работы с Argo CD можно использовать не только веб-интерфейс и кастомные ресурсы, но и CLI-утилиту `argocd`. Бинарный файл можно загрузить из раздела «Documentation» веб-интерфейса Argo CD. Инструкцию по работе с CLI-утилитой можно получить, выполнив `argocd --help`.

Развёртывание нескольких экземпляров Argo CD

Если в кластере требуется несколько экземпляров Argo CD, создайте для каждого из них отдельный неймспейс (или проект DKP) и отдельный объект ArgoCD.

Например, создайте:

- отдельный экземпляр для production-окружения;
- отдельный экземпляр для тестовых окружений;
- отдельный экземпляр для конкретной команды или проекта.

При таком подходе каждый экземпляр Argo CD управляется независимо и имеет собственную конфигурацию.

⚠ В одном неймспейсе поддерживается создание не более одного объекта ArgoCD.

Расширенные настройки

Включение режима высокой доступности

Чтобы включить режим высокой доступности, установите параметр `spec.ha.enabled: true` в объекте ArgoCD.

Пример:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  ha:
    enabled: true
    resources:
      requests:
        cpu: "500m"
        memory: "512Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
```

⚠ Режим высокой доступности обеспечивает высокую доступность только для хранилища состояния Redis с использованием HAProxy. Он не делает все компоненты Argo CD высокодоступными автоматически.

i Для запуска Argo CD в режиме высокой доступности требуется не менее трёх узлов кластера из-за правил `pod anti-affinity`. Кластеры, работающие только с IPv6, не поддерживаются.

При включённом режиме высокой доступности изменения в `.spec.redis.resources` не применяются. Ограничения и запросы ресурсов для Redis настраивайте через параметр `.spec.ha.resources`.

Предоставление доступа к кластерным ресурсам

По умолчанию экземпляр Argo CD получает привилегии только на неймспейс, в котором он запущен, и неймспейсы, помеченные лейблом `argocd.argoproj.io/managed-by`. Значение лейбла должно совпадать с названием неймспейса, в котором запущен экземпляр Argo CD.

Чтобы разрешить создание cluster-wide-ресурсов, укажите в параметре `clusterConfigNamespaces` настроек модуля `operator-argo` имя неймспейса, в котором запущен целевой экземпляр Argo CD.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-argo
spec:
  enabled: true
  settings:
    clusterConfigNamespaces: argocd
  version: 1
```

Если экземпляров Argo CD несколько, перечислите их в параметре `clusterConfigNamespaces` через запятую.

После внесения изменений в `ModuleConfig` автоматически будут созданы следующие кластерные роли (ClusterRole) и кластерные биндинги (ClusterRoleBinding):

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - impersonate
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd-application-controller
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller
```

```

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-application-controller
subjects:
- kind: ServiceAccount
  name: argocd-argocd-application-controller
  namespace: argocd
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-server
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - get
  - delete
  - patch
- apiGroups:
  - argoproj.io
  resources:
  - applications
  - applicationsets
  verbs:
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - list
- apiGroups:
  - batch
  resources:
  - jobs
  - cronjobs
  - cronjobs/finalizers
  verbs:
  - create
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd-server

```

```

app.kubernetes.io/part-of: argocd
name: argocd-argocd-argocd-server
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-server
subjects:
- kind: ServiceAccount
  name: argocd-argocd-server
  namespace: argocd

```

i Если полномочия, описанные в кластерных ролях по умолчанию, избыточны, в настройках объекта ArgoCD укажите `spec.defaultClusterScopedRoleDisabled: true`. В этом случае кластерные роли не будут созданы автоматически, и вы сможете самостоятельно описать необходимый уровень привилегий для ServiceAccount, используемого экземпляром Argo CD.

i При необходимости уровень доступа к кластерным ресурсам можно переопределить на уровне объекта AppProject, используя параметры `spec.clusterResourceBlacklist` и `spec.clusterResourceWhitelist`.

Использование собственного домена кластера

Если в кластере используется домен, отличный от `cluster.local`, укажите его в параметре `spec.clusterDomain` объекта ArgoCD.

Пример:

```

apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  clusterDomain: prod.local

```

Дополнительные ресурсы

- Официальный сайт Argo CD
- Официальный сайт Argo CD Operator
- Документация модуля `operator-argo`

Настройка аутентификации и авторизации

Argo CD поддерживает локальную аутентификацию, а также интегрирован с подсистемой идентификации и доступа Deckhouse Kubernetes Platform. В документации DKP можно подробнее узнать о настройке аутентификации и авторизации.

 Если в объекте ArgoCD не заданы дополнительные настройки, по умолчанию активна локальная учётная запись `admin` с ролью `admin`.

Об управлении полномочиями пользователей и групп можно узнать в разделе «Настройка ролевой модели доступа».

Локальная аутентификация

При создании объекта ArgoCD автоматически создаётся пользователь `admin` с ролью `admin`. Пароль пользователя `admin` генерируется автоматически. Чтобы получить его, выполните команду:

```
d8 k -n argocd get secret argocd-cluster -o jsonpath='{.data.admin\.password}' | base64 -d
```

Создание дополнительных локальных пользователей

При создании локального пользователя можно определить, будут ли у него права на доступ к веб-интерфейсу Argo CD (атрибут `login`) и/или к API Argo CD (атрибут `apiKey`).

Чтобы создать локальных пользователей, задайте параметр `spec.localUsers` ArgoCD, например:

```
d8 k create -f - <<EOF
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  localUsers:
    # Пользователь "deploy" с правами на вход в веб-интерфейс и генерацию токенов.
    - name: deploy
      apiKey: true
      login: true
      enabled: true
    # Пользователь "ci-bot" только для автоматизации, без доступа к веб-интерфейсу.
    - name: ci-bot
      apiKey: true
      login: false
      enabled: true
EOF
```

После создания пользователя и применения манифеста объекта ArgoCD сгенерируйте пароль для доступа к веб-интерфейсу, если пользователю выданы соответствующие права.

Чтобы задать пароль через Secret, сначала вычислите его `bcrypt`-хеш, а затем закодируйте результат в Base64:

```
ARGOCD_USER=<USER_NAME>
ARGOCD_PASS=$(echo -n '<USER_PASSWORD>' | httpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0)
d8 k -n argocd patch secret argocd-secret -p "{\"data\": {\"accounts.$ARGOCD_USER.password\": \"\$ARGOCD_PASS\"}}"
```

i

Пароль локального пользователя можно задать или изменить с помощью CLI-утилиты `argocd`:

```
argocd login <ARGOCD_DOMAIN>:443 --username admin --password <ADMIN_PASSWORD>
argocd account update-password \
  --account <ACCOUNT> \
  --current-password <ADMIN_PASSWORD> \
  --new-password <NEW_PASSWORD>
```

Создание пользовательского токена

Для работы с API у пользователя должны быть соответствующие права (параметр `apiKey`, пример в разделе выше). Чтобы выпустить токен, используйте веб-интерфейс Argo CD или CLI-утилиту `argocd`.

Чтобы выпустить токен через веб-интерфейс, перейдите в раздел «Settings» → «Accounts», выберите нужного пользователя и нажмите «Generate New» в разделе «Tokens».

Чтобы выпустить токен с помощью CLI-утилиты `argocd`, выполните команды (укажите необходимые значения):

```
argocd login <ARGOCD_DOMAIN>:443 --username admin
argocd account generate-token --account <ACCOUNT>
```

i Если у пользователя нет прав на доступ к веб-интерфейсу (`login`), токен для него может выпустить другой пользователь с правами администратора.

Отключение локальной аутентификации

Чтобы отключить локальную аутентификацию, отключите пользователя `admin`, задав параметру `spec.disableAdmin` ArgoCD значение `true`. Также удалите всех дополнительных локальных пользователей, если они были созданы ранее (подробнее — в разделе «Создание дополнительных локальных пользователей»).

После отключения локальной аутентификации проверьте правила доступа в разделе «Настройка ролевой модели доступа».

Аутентификация с помощью SSO

Перед настройкой объекта ArgoCD создайте объект `DexClient`, который используется в качестве OAuth2-клиента. Он необходим для интеграции с Deckhouse Kubernetes Platform:

```
d8 k create -f -<<EOF
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: argocd
  namespace: argocd
spec:
  redirectURIs:
    - https://<ARGOCD_DOMAIN>/api/dex/callback
    - https://<ARGOCD_DOMAIN>/api/dex/callback-reserve
EOF
```

`<ARGOCD_DOMAIN>` — это полное доменное имя (Fully Qualified Domain Name, FQDN), заданное в секции `.spec.server.host` ArgoCD.

Дождитесь, пока Deckhouse Kubernetes Platform создаст `Secret` с секретным ключом для клиента:

```
d8 k -n argocd get secret/dex-client-argocd
```

Настройте объект ArgoCD на использование SSO в Deckhouse Kubernetes Platform:

```

apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    scopes: "[groups, email]"
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
        config:
          issuer: "https://dex.<CLUSTER_DOMAIN>/"
          clientId: "dex-client-argocd@argocd"
          clientSecret: "$dex-client-argocd:clientSecret"
          insecureEnableGroups: true
          scopes:
            - profile
            - email
            - openid
            - groups
      provider: dex
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      ingressClassName: <INGRESS_CLASS_NAME>
    tls:
      - hosts:
          - <ARGOCD_DOMAIN>
        secretName: argocd-ingress-tls
    insecure: true

```

Перезапустите сервер Argo CD:

```
d8 k -n argocd rollout restart deploy/argocd-server
```



Если не перезапустить сервер Argo CD, попытка входа завершится ошибкой, а в журнале сервера Argo CD появится сообщение об ошибке (issue [argoproj/argo-cd#13526](#)).

Пример сообщения об ошибке...

```

time="2024-10-16T14:12:59Z" level=warning msg="Failed to verify token: failed to verify token:
token verification failed for all audiences: error for aud \"argo-cd\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>, error for aud \"argo-cd-cli\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>"

```

Использование самоподписного сертификата

Предварительно получите самоподписанный сертификат, используемый подсистемой идентификации и доступа Deckhouse Kubernetes Platform:

```
d8 k -n d8-user-authn get secret ingress-tls -o jsonpath='{.data.tls\.crt}' | base64 -d
```

Затем добавьте полученный сертификат в конфигурацию OIDC-коннектора, указав его в секции `rootCAs` `ArgoCD`:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
            config:
              issuer: "https://dex.<CLUSTER_DOMAIN>/"
              rootCAs:
                - |
                  -----BEGIN CERTIFICATE-----
                  <Самоподписанный сертификат, полученный на предыдущем шаге>
                  -----END CERTIFICATE-----
              clientId: "dex-client-argocd@argocd"
              clientSecret: "$dex-client-argocd:clientSecret"
              insecureEnableGroups: true
              scopes:
                - profile
                - email
                - openid
                - groups
        provider: dex
    server:
      host: <ARGOCD_DOMAIN>
      ingress:
        enabled: true
        tls:
          - hosts:
              - <ARGOCD_DOMAIN>
            secretName: argocd-ingress-tls
      insecure: true
```

Создание пользовательского токена

Argo CD не позволяет выпускать бессрочные токены для пользователей, аутентифицированных с помощью SSO. При этом такие пользователи могут использовать CLI-утилиту `argocd`, указав при аутентификации флаг `--sso`:

```
argocd login <ARGOCD_DOMAIN>:443 --sso
```

При выполнении этой команды на рабочей станции администратора будет запущен веб-браузер с формой аутентификации в Deckhouse Kubernetes Platform.



Чтобы выпустить бессрочный токен, создайте локального пользователя Argo CD и укажите для него атрибут `apiKey`. В этом случае у пользователя будут права доступа только к Argo CD API. Подробнее — в разделе «Создание дополнительных локальных пользователей».

Дополнительные ресурсы

- Официальный сайт Argo CD
- Официальный сайт Argo CD Operator

- Кастомные ресурсы модуля `operator-argo`
- Кастомные ресурсы модуля `user-authn`

Настройка ролевой модели доступа

Argo CD использует собственную ролевую модель (Role-based Access Control, RBAC), не основанную на ролевой модели Kubernetes и Deckhouse Kubernetes Platform. Ролевая модель Argo CD позволяет ограничивать доступ к ресурсам и операциям через собственные политики и роли.

Перед настройкой ролевой модели доступа выполните настройку аутентификации и авторизации. После этого назначайте роли пользователям и группам, а также задавайте разрешения на уровне всего экземпляра Argo CD или отдельных проектов с помощью объекта AppProject.

Правила ролевой модели доступа (Role-Based Access Control, RBAC) можно определить в двух местах:

- глобально — в объекте ArgoCD;
- на уровне проекта — в ролях объекта AppProject.

Встроенные роли

В Argo CD есть две предопределённые роли с набором политик:

- `role:readonly` — доступ только на чтение;
- `role:admin` — доступ с полномочиями администратора.

Полное описание политик предопределённых ролей...

```
# Политики роли readonly.
p, role:readonly, applications, get, /*/, allow
p, role:readonly, applicationsets, get, /*/, allow
p, role:readonly, certificates, get, /*/, allow
p, role:readonly, clusters, get, /*/, allow
p, role:readonly, repositories, get, /*/, allow
p, role:readonly, write-repositories, get, /*/, allow
p, role:readonly, projects, get, /*/, allow
p, role:readonly, accounts, get, /*/, allow
p, role:readonly, gpgkeys, get, /*/, allow
p, role:readonly, logs, get, /*/, allow

# Политики роли admin.
p, role:admin, applications, create, /*/, allow
p, role:admin, applications, update, /*/, allow
p, role:admin, applications, update/*, /*/, allow
p, role:admin, applications, delete, /*/, allow
p, role:admin, applications, delete/*, /*/, allow
p, role:admin, applications, sync, /*/, allow
p, role:admin, applications, override, /*/, allow
p, role:admin, applications, action/*, /*/, allow
p, role:admin, applicationsets, get, /*/, allow
p, role:admin, applicationsets, create, /*/, allow
p, role:admin, applicationsets, update, /*/, allow
p, role:admin, applicationsets, delete, /*/, allow
p, role:admin, certificates, create, /*/, allow
p, role:admin, certificates, update, /*/, allow
p, role:admin, certificates, delete, /*/, allow
p, role:admin, clusters, create, /*/, allow
p, role:admin, clusters, update, /*/, allow
p, role:admin, clusters, delete, /*/, allow
p, role:admin, repositories, create, /*/, allow
p, role:admin, repositories, update, /*/, allow
p, role:admin, repositories, delete, /*/, allow
p, role:admin, write-repositories, create, /*/, allow
p, role:admin, write-repositories, update, /*/, allow
p, role:admin, write-repositories, delete, /*/, allow
p, role:admin, projects, create, /*/, allow
p, role:admin, projects, update, /*/, allow
p, role:admin, projects, delete, /*/, allow
p, role:admin, accounts, update, /*/, allow
p, role:admin, gpgkeys, create, /*/, allow
p, role:admin, gpgkeys, delete, /*/, allow
p, role:admin, exec, create, /*/, allow

# Роль admin включает все полномочия роли readonly.
g, role:admin, role:readonly

# Локальному пользователю admin назначена роль admin.
g, admin, role:admin
```

Политика по умолчанию для аутентифицированных пользователей

После успешной аутентификации пользователь получает роль, указанную в параметре `spec.rbac.defaultPolicy` ArgoCD.



Все аутентифицированные пользователи получают как минимум те разрешения, которые заданы в параметре `spec.rbac.defaultPolicy`. Эти права нельзя отозвать правилом с эффектом `deny`.

Рекомендуется создать отдельную роль, например `role:authenticated`, выдать ей минимальный набор разрешений и использовать её в `spec.rbac.defaultPolicy`.

Пример:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    defaultPolicy: role:authenticated
    policy: |
      p, role:authenticated, applications, get, */*, allow
  ...
```

Анонимный доступ

Если включить анонимный доступ к экземпляру Argo CD, пользователи смогут получить права согласно политике, указанной в параметре `spec.rbac.defaultPolicy`, без аутентификации.

Анонимный доступ включается параметром `spec.usersAnonymousEnabled` ArgoCD.



При включении анонимного доступа создайте отдельную роль по умолчанию, например `role:unauthenticated`, и назначьте её в `spec.rbac.defaultPolicy`.

Структура ролевой модели доступа

Синтаксис ролевой модели доступа в Argo CD основан на модели Casbin, в которой используются два типа записей:

- привязка пользователя или группы к роли;
- назначение разрешений роли, пользователю или группе.

Привязка к роли

Синтаксис:

```
g, <user/group>, <role>
```

Где:

- `<user/group>` — локальный пользователь, пользователь SSO или группа;
- `<role>` — внутренняя роль Argo CD.

Пример:

```
g, my-org:team-beta, role:admin
g, user@example.org, role:readonly
```

Политика доступа

Синтаксис:

```
p, <role/user/group>, <resource>, <action>, <object>, <effect>
```

Где:

- `<role/user/group>` — субъект, которому назначается правило;
- `<resource>` — тип ресурса;
- `<action>` — разрешённая операция;

- `<object>` — объект, к которому применяется правило;
- `<effect>` — результат проверки: `allow` или `deny`.

Пример:

```
p, role:developer, applications, sync, dev-project/*, allow
p, role:developer, logs, get, dev-project/*, allow
```

i Чтобы назначать правила группе, сначала привяжите группу к роли с помощью записи `g, <group>, <role>`. После этого назначайте разрешения самой роли.

Ресурсы и поддерживаемые действия

Ниже перечислены основные ресурсы Argo CD и действия, которые можно использовать в политиках RBAC.

Ресурс	get	create	update	delete	sync	action	override	invoke
applications	☐	☐	☐	☐	☐	☐	☐	☐
applicationsets	☐	☐	☐	☐	☐	☐	☐	☐
clusters	☐	☐	☐	☐	☐	☐	☐	☐
projects	☐	☐	☐	☐	☐	☐	☐	☐
repositories	☐	☐	☐	☐	☐	☐	☐	☐
accounts	☐	☐	☐	☐	☐	☐	☐	☐
certificates	☐	☐	☐	☐	☐	☐	☐	☐
gpgkeys	☐	☐	☐	☐	☐	☐	☐	☐
logs	☐	☐	☐	☐	☐	☐	☐	☐
exec	☐	☐	☐	☐	☐	☐	☐	☐
extensions	☐	☐	☐	☐	☐	☐	☐	☐

Политики, привязанные к приложениям

Некоторые ресурсы привязаны к конкретным приложениям:

- `applications`;
- `applicationsets`;
- `logs`;
- `exec`.

Для таких ресурсов значение `<object>` обычно имеет формат `<APP_PROJECT>/<APP_NAME>`.

Пример:

```
p, example-user, applications, get, *, allow
p, example-user, logs, get, example-project/my-app, allow
```

Если в Argo CD включён режим размещения приложений в произвольных неймспейсах, формат `<object>` меняется на `<APP_PROJECT>/<APP_NAMESPACE>/<APP_NAME>`.

Пример:

```
p, example-user, applications, get, */app-namespace/*, allow
p, example-user, logs, get, example-project/app-namespace/my-app, allow
```

Ресурс `applications`

Ресурс `applications` поддерживает как базовые действия, так и более детальные правила.

Детализированные разрешения для `update` и `delete`

Разрешения `update` и `delete`, выданные на приложение, позволяют изменять или удалять само приложение, но не его вложенные ресурсы.

Чтобы разрешить операцию над ресурсами приложения, используйте формат:

```
<action>/<group>/<kind>/<namespace>/<name>
```

Например, чтобы разрешить пользователю удалять только Pod в приложении `prod-app`:

```
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

Чтобы разрешить обновление всех ресурсов приложения, но не самого приложения:

```
p, example-user, applications, update/*, default/prod-app, allow
```

Чтобы запретить удаление приложения, но разрешить удаление Pod:

```
p, example-user, applications, delete, default/prod-app, deny
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

Чтобы разрешить обновление приложения, но запретить обновление любых субресурсов:

```
p, example-user, applications, update, default/prod-app, allow
p, example-user, applications, update/*, default/prod-app, deny
```



Argo CD не использует символ `/` как разделитель при сравнении glob-шаблонов. Поэтому рекомендуется всегда указывать полный путь ресурса, то есть использовать шаблоны с четырьмя символами `/`.

Действие `action`

Действие `action` соответствует встроенным или пользовательским действиям над ресурсами приложения.

Формат значения `<action>`:

```
action/<group>/<kind>/<action-name>
```

Например:

- `action/extensions/DaemonSet/restart` — действие для `DaemonSet`;
- `action//Pod/maintenance-off` — действие для `Pod`, у которого нет API-группы.

Пример политики:

```
p, example-user, applications, action//Pod/maintenance-off, default/*, allow
p, example-user, applications, action/extensions/DaemonSet/*, default/*, allow
```

Чтобы разрешить все действия над ресурсами приложения:

```
p, example-user, applications, action/*, default/*, allow
```

Действие `override`

Разрешение `override` позволяет передавать произвольные манифесты или другую ревизию при синхронизации `Application`.



Право `override` позволяет пользователю фактически изменить состав или состояние развёрнутых ресурсов приложения. Выдавайте его только тем пользователям, которым это действительно необходимо.

Если в ConfigMap `argocd-cm` включён параметр

`application.sync.requireOverridePrivilegeForRevisionSync: 'true'`, передача произвольной ревизии при синхронизации также будет требовать право `override`.

Другие ресурсы

Ресурс `applicationsets`

Ресурс `applicationsets` также относится к политикам, привязанным к приложениям. Разрешение `create` для этого ресурса фактически даёт возможность создавать объекты `Application` через объект `ApplicationSet`.

Пример:

```
p, dev-group, applicationsets, *, dev-project/*, allow
```

С таким правилом пользователи из `dev-group` смогут создавать `ApplicationSet`, который управляет приложениями только в рамках проекта `dev-project`.

Ресурс `logs`

Разрешение `get` для ресурса `logs` позволяет просматривать логи Pod приложения через веб-интерфейс Argo CD. По смыслу это эквивалентно `d8 k logs`.

Ресурс `exec`

Разрешение `create` для ресурса `exec` позволяет открывать `exec`-сессию в Pod приложения через веб-интерфейс Argo CD. По смыслу это близко к `d8 k exec`.

Ресурс `extensions`

Ресурс `extensions` используется для вызова проху-расширений. Проверка таких правил работает совместно с ресурсом `applications`: пользователь должен иметь право на чтение приложения, из которого выполняется запрос.

Пример:

```
p, example-user, applications, get, default/*, allow
p, example-user, extensions, invoke, httpbin, allow
```

Эффект `deny`

Если политика с эффектом `deny` совпала с запросом, доступ будет запрещён, даже если есть более специфичные правила с эффектом `allow`.

Порядок строк в `spec.rbac.defaultPolicy` на результат не влияет.

Порядок проверки политик и режимы сопоставления

Проверка доступа выполняется в два этапа:

1. Проверяются правила из `spec.rbac.defaultPolicy`.
2. Если решение не найдено, проверяются правила, относящиеся к пользователю и его группам.

Если доступ явно разрешён или запрещён политикой по умолчанию, дальнейшая проверка не выполняется.

Argo CD поддерживает два режима сопоставления значений, задаваемых в `spec.rbac.policyMatchMode`:

- `glob` — сопоставление по `glob`-шаблонам;
- `regex` — сопоставление по регулярным выражениям.

Сопоставление в режиме `glob`

В режиме `glob` токены рассматриваются как цельные строки без специальных разделителей.

Пример политики:

```
p, example-user, applications, action/extensions/*, default/*, allow
```

Если пользователь `example-user` вызывает действие `extensions/DaemonSet/test`, будут выполнены такие проверки:

1. `example-user` совпадает с `example-user`.
2. `applications` совпадает с `applications`.
3. `action/extensions/DaemonSet/test` совпадает с `action/extensions/*`.
4. `default/my-app` совпадает с `default/*`.

Использование пользователей и групп SSO

Параметр `scopes` определяет, какие OIDC scopes Argo CD должен анализировать при проверке RBAC помимо `sub`. Если параметр не задан, по умолчанию используется значение `'[groups]'`.

Пример объекта ArgoCD с настройкой `scopes`:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    policy: |
      p, my-org:team-alpha, applications, sync, my-project/*, allow
      g, my-org:team-beta, role:admin
      g, user@example.org, role:admin
      g, admin, role:admin
      g, role:admin, role:readonly
    defaultPolicy: role:readonly
    scopes: '[groups, email]'
```

В этом примере:

1. `g, admin, role:admin` явно привязывает встроенного пользователя `admin` к роли `role:admin`.
2. `g, role:admin, role:readonly` задаёт наследование ролей: все, у кого есть `role:admin`, автоматически получают и права `role:readonly`.

Этот подход можно комбинировать с ролями на уровне AppProject. Например, можно создать проект `team-beta-project` и назначить права пользователям и группам:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-beta-project
  namespace: argocd
spec:
  roles:
    - name: admin
      description: Права администратора для team-beta.
      policies:
        - p, proj:team-beta-project:admin, applications, *, team-beta-project/*, allow
      groups:
        - user@example.org
        - my-org:team-beta
```

Локальные пользователи

Локальным пользователям можно выдать доступ двумя способами:

- назначить правила напрямую;
- привязать пользователя к роли.

Пример назначения правила напрямую:

```
p, my-local-user, applications, sync, my-project/*, allow
```

Пример привязки к роли:

```
g, my-local-user, role:admin
```



Если одновременно используются SSO и локальные пользователи, возможна неоднозначность. Например, если локальный пользователь `sally` привязан к `role:admin`, а значение одного из SSO scopes тоже равно `sally`, такой пользователь SSO также может получить права `role:admin`.

Чтобы избежать этой ситуации, при одновременном использовании SSO и локальных пользователей рекомендуется назначать локальным пользователям правила напрямую, а не через роли.

Пример:

```
p, my-local-user, *, *, *, allow
```

Тестирование политик RBAC

Чтобы проверить корректность правил RBAC, используйте CLI-утилиту `argocd`. Чтобы определить, может ли конкретный пользователь, группа или роль выполнить определённое действие, используйте команду:

```
argocd admin settings rbac can
```

Например, выполните команду ниже, чтобы проверить, есть ли у пользователя `admin@deckhouse.io` права на создание `applications` в проекте `default`:

```
argocd admin settings rbac can admin@deckhouse.io create application 'default/*' --namespace argocd
```

Дополнительные ресурсы

- Официальный сайт Argo CD
- Официальный сайт Argo CD Operator
- Кастомные ресурсы модуля `operator-argo`

Marketplace

Marketplace — это система управления единицами поставки Deckhouse Kubernetes Platform (DKP) (Packages). Позволяет администраторам подключать реестры пакетов, обнаруживать доступные пакеты и открывать пользователям проектов возможность их установки.



Marketplace доступен начиная с DKP версии 1.76.

Задачи администратора

Администратор кластера:

- Подключает реестры пакетов (создавая объекты `PackageRepository`).
- Следит за операциями сканирования, которые обнаруживают пакеты в реестре.
- Обеспечивает пользователям доступ к версиям пакетов для установки в свои неймспейсы.

Пользователи взаимодействуют с пакетами через объект `Application` (подробнее в разделе [Использование](#) → [Marketplace](#)).

Ключевые ресурсы

Ресурс	Короткое имя	Область	Описание
<code>PackageRepository</code>	—	Cluster	Исходный реестр пакетов
<code>PackageRepositoryOperation</code>	<code>pro</code>	Cluster	Операция сканирования репозитория
<code>ApplicationPackageVersion</code>	<code>apv</code>	Cluster	Обнаруженная версия пакета
<code>ApplicationPackage</code>	—	Cluster	Агрегированные метаданные пакета

Ресурс	Короткое имя	Область	Описание
Application	app	Namespace	Установленный экземпляр приложения (управляется пользователями)

В разделе «Репозитории пакетов» описано подключение реестра и проверка его статуса.

В разделе «Сканирование» описан мониторинг операций сканирования и запуск сканирования вручную.

Дополнительные ресурсы

- Использование Marketplace
- Глобальные кастомные ресурсы

Репозитории пакетов

Подключение Deckhouse Kubernetes Platform (DKP) к container registry, содержащему пакеты приложений, выполняется с помощью PackageRepository. После подключения DKP автоматически сканирует реестр и создаёт объекты ApplicationPackageVersion для каждой обнаруженной версии пакета.

Пример манифеста PackageRepository:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepository
metadata:
  name: my-registry
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <base64-encoded-docker-config>
```

Управление аутентификацией и интервалом сканирования

Аутентификация

Для аутентификации в реестре может использоваться один из следующих способов:

- **dockerCfg**: Docker-конфиг в формате base64 (формат `~/.docker/config.json`). Предпочтителен, если реестр использует токен-based аутентификацию.
- **login + password**: явные учётные данные.

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    login: my-user
    password: my-password
```

Если реестр использует самоподписанный TLS-сертификат, передайте его через параметр **ca**:

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <base64-encoded-docker-config>
    ca: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

Интервал сканирования

По умолчанию DKP пересканирует реестр каждые **6 часов**. Переопределить интервал можно с помощью параметра `scanInterval`:

```
spec:
  registry:
    repo: registry.example.com/packages
  scanInterval: 1h30m
```

Проверка состояния репозитория

Состояние репозитория отображается в статусе объекта `PackageRepository`.

Для вывода краткой информации о статусе, используйте следующую команду:

```
d8 k get packagerepository <REPOSITORY_NAME>
```

Колонки вывода:

Колонка	Описание
Phase	Текущее состояние репозитория
Scan	Время последнего сканирования
MSG	Сообщение из условия последнего сканирования
Packages	Общее количество обнаруженных пакетов (скрыто по умолчанию, используйте <code>-o wide</code>)

Для получения детальной информации о статусе, используйте следующую команду:

```
d8 k get packagerepository <REPOSITORY_NAME> -o yaml
```

Ключевые поля статуса:

Поле	Описание
<code>status.phase</code>	Текущая фаза репозитория
<code>status.lastScanTime</code>	Время последнего сканирования с любым результатом
<code>status.lastChangeTime</code>	Время последнего сканирования, которое нашло хотя бы одну новую версию
<code>status.lastNewVersions</code>	Количество новых версий, найденных при последнем сканировании
<code>status.packagesCount</code>	Общее число пакетов в репозитории
<code>status.packages[]</code>	Список пакетов с полями <code>name</code> и <code>type</code>
<code>status.conditions</code>	Детальные условия, включая <code>LastScanSucceeded</code>

Условие `LastScanSucceeded`:

```
d8 k get packagerepository my-registry \
-o jsonpath='{.status.conditions[?(@.type=="LastScanSucceeded")].message}'
```

Просмотр обнаруженных версий пакетов

После успешного сканирования в кластере появляются объекты `ApplicationPackageVersion` (можно использовать сокращенное имя — `apv`):

```
d8 k get apv
```

Пример вывода:

NAME	PACKAGE	REPOSITORY	TRANSITIONTIME	METADATALOADED
MESSAGE USED BY				
my-registry-redis-v7.2.0	redis	my-registry	5m	True
my-registry-postgres-v15.0.0	postgres	my-registry	5m	True

Фильтрация по имени пакета, можно использовать следующую команду (в примере фильтруются версии пакета `redis`):

```
d8 k get apv -l package=redis
```

i MetadataLoaded=True означает, что OpenAPI-схема пакета, описание и требования успешно загружены из реестра. Пакет с MetadataLoaded=False не может быть установлен до получения метаданных.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Сканирование

Deckhouse Kubernetes Platform (DKP) использует PackageRepositoryOperation для сканирования реестров пакетов. Каждая операция сканирования обнаруживает новые версии пакетов и создаёт или обновляет объекты ApplicationPackageVersion. Операции создаются автоматически по расписанию или могут создаваться вручную при необходимости.

Просмотр операций сканирования

Для просмотра всех операций сканирования используйте команду (можно использовать `pro` — сокращенное имя для `packagerepositoryoperations`):

```
d8 k get pro
```

Пример вывода:

NAME	COUNT	COMPLETED	MSG	COMPLETIONTIME
test-scan-1780052895	23	True		3h38m
test-scan-1780053890	23	True		3h22m

Колонки вывода:

Колонка	Описание
Count	Общее количество пакетов, найденных при сканировании
Completed	Завершена ли операция (True / False)
MSG	Сообщение из условия Completed
CompletionTime	Время завершения операции

Для фильтрации операций по конкретному репозиторию используйте следующую команду:

```
d8 k get pro -l packages.deckhouse.io/repository=<REPO_NAME>
```

Детали операции сканирования

Для получения полных результатов сканирования с детализацией по пакетам используйте следующую команду:

```
d8 k get pro <имя-операции> -o yaml
```

Ключевые поля статуса:

Поле	Описание
<code>status.startTime</code>	Время начала операции
<code>status.completionTime</code>	Время завершения операции
<code>status.packages.total</code>	Всего найдено пакетов
<code>status.packages.processedOverall</code>	Всего успешно обработано пакетов
<code>status.packages.newVersionsOverall</code>	Суммарное количество новых версий по всем пакетам
<code>status.packages.processed[]</code>	Результаты по каждому пакету: <code>name</code> , <code>type</code> , <code>foundVersions</code> , <code>newVersions</code>
<code>status.packages.failed[]</code>	Пакеты с ошибками: <code>name</code> , <code>errors[]</code> с полями <code>version</code> и <code>message</code>
<code>status.packages.discovered[]</code>	Пакеты, впервые обнаруженные в этой операции

Пример команды для просмотра пакетов с ошибками:

```
d8 k get pro <имя-операции> \
  -o jsonpath='{range .status.packages.failed[*]}.{name}: {range .errors[*]}.{version} -
  {message}}{"\n"}{end}{end}'
```

Запуск сканирования вручную

По умолчанию каждый `PackageRepository` создаёт новую операцию сканирования через **6 часов** после завершения предыдущей (настраивается через `spec.scanInterval`).

Для немедленного сканирования создайте `PackageRepositoryOperation` вручную, по следующему примеру:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepositoryOperation
spec:
  packageRepositoryName: my-registry
  type: Update
  update:
    fullScan: true
    timeout: 5m
```

i Используйте `generateName` вместо `name`, чтобы создавать несколько операций без конфликтов имён.

Альтернативно, для создания операции сканирования, можно использовать следующую команду:

```
d8 system package scan <REPO_NAME>
```

Эта команда создаёт `PackageRepositoryOperation` с `spec.type: Update` и `spec.update.fullScan: true`.

Флаг `fullScan`

Значение	Поведение
<code>true</code>	Повторно проверяет все теги в реестре, включая уже известные
<code>false</code> (по умолчанию)	Обрабатывает только теги, добавленные с момента последнего сканирования (инкрементальное)

Используйте `fullScan: true`, если вы подозреваете, что реестр был изменён вне обычного рабочего процесса, или когда в объектах `ApplicationPackageVersion` отсутствуют версии, которые есть в реестре.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Обновление платформы

В Deckhouse Kubernetes Platform (DKP) реализован механизм управления обновлениями самой платформы и компонентов Kubernetes-кластера. Для последовательного и безопасного внедрения новых версий используется система из пяти каналов обновлений: от самых свежих и нестабильных (Alpha) до максимально проверенных (Rock Solid). Новая версия проходит через каналы поэтапно для выявления проблем на ранних стадиях и обеспечения стабильности production-сред. Подробнее о каждом из каналов в разделе Архитектура.



DKP не поддерживает понижение версии (downgrade). После установки релиза откат платформы на более раннюю минорную или патч-версию невозможен.

При переключении на более стабильный канал платформа остаётся на текущем релизе до появления более новой версии на этом канале.

Откат на одну минорную версию назад возможен только для версии Kubernetes (control plane), а не для версии DKP.

Процесс обновления можно настроить: доступны автоматический и ручной режимы, задание временных окон и другие параметры — подробнее в разделе Настройка обновлений.

Для контроля состояния обновлений и оперативного получения информации об ошибках используется система алертов — подробнее в разделе Настройка уведомлений.

Поддерживаются:

- Получение уведомлений о релизах — отправка оповещений о новых релизах через вебхуки.
- Получение Changelog — каждая версия DKP сопровождается списком изменений, который доступен как в кластере, так и в уведомлении о новом релизе.
- Проверка зависимостей при обновлении — проверяет зависимости перед обновлением и предотвращает возможные конфликты между компонентами.

С обновлением DKP в закрытом окружении можно ознакомиться подробнее в руководстве «Обновление DKP в закрытом окружении».

Настройка обновлений

Deckhouse Kubernetes Platform (DKP) поддерживает гибкий механизм обновлений. Вы можете выбрать канал обновлений и настроить режим установки новых версий. Каналы обновлений помогают сбалансировать скорость получения новых функций и стабильность.

Настроив режим обновлений, можно включить автоматическую установку новых версий или управлять ею вручную, а также задать окна обновлений — допустимые интервалы времени для выполнения обновлений. Такая настройка исключает установку в неподходящее время и обеспечивает контроль над переходом на новые релизы.



Актуальная информация о версиях DKP на разных каналах обновлений доступна на сайте releases.deckhouse.ru.

Проверка текущего канала обновлений

Чтобы узнать, какой канал обновлений используется в кластере, выполните следующую команду:

```
d8 k get mc deckhouse -o yaml | grep releaseChannel
```

Пример вывода:

```
releaseChannel: Stable
```

Смена канала обновлений

Чтобы сменить канал обновлений, укажите его название в параметре `settings.releaseChannel` модуля `deckhouse`.

Пример конфигурации с каналом `Stable`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
```

Режимы обновления

DKP поддерживает два режима обновления, которые определяют порядок перехода на новую версию:

- **Автоматический режим** — кластер обновляется сразу после появления новой версии на используемом канале обновлений. Доступно два варианта настройки:
 - `AutoPatch` — устанавливаются обновления патч-версий в рамках текущей минорной версии;
 - `Auto` — устанавливаются и патч-версии, и минорные версии.
- **Ручной режим (Manual)** — и патч-версии, и минорные версии применяются после ручного подтверждения.

Для автоматического режима доступны настройки окна обновлений — разрешённые обновления применяются с учётом `update.windows`, если окна заданы. Если окна не заданы, обновление выполняется сразу после появления версии на канале.

i

Настройки `update` модуля `deckhouse` (режим и окна) по умолчанию применяются и к встроенным, и к внешним модулям.

Если у модуля нет собственной политики обновления (`ModuleUpdatePolicy`) и в его `ModuleConfig` не указан параметр `updatePolicy`, модуль наследует `releaseChannel` и `update` из `ModuleConfig deckhouse`.

Только патч-версии в рамках текущей минорной (AutoPatch)

Чтобы DKP обновлялся в пределах текущей минорной версии (устанавливались патч-версии), используйте режим `AutoPatch`.

Например, при установленной версии `v1.70.1` DKP сможет автоматически обновиться до `v1.70.2`, но не перейдёт на `v1.71.*` без ручного подтверждения.

Это значение задано по умолчанию. Чтобы явно установить режим `AutoPatch`, выполните команду:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"AutoPatch"}}}}'
```

Чтобы подтвердить обновление минорной версии, выполните следующую команду, указав необходимую версию DKP вместо `<DECKHOUSE-VERSION>`:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

Автоматическое обновление всех версий (Auto)

В режиме `Auto` DKP автоматически применяет и патч-версии, и минорные версии с учётом окон обновлений, если они заданы.

Чтобы включить режим `Auto`, выполните команду:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Auto"}}}}'
```

Ручной режим (Manual)

В режиме `Manual` DKP получает информацию о новых версиях в кластер, но применение и патч-версий, и минорных версий требует ручного подтверждения.

Чтобы включить режим `Manual`, выполните команду:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
```

Проверка текущего режима обновления

Чтобы узнать, какой режим обновления используется в кластере, проверьте конфигурацию модуля `deckhouse` с помощью следующей команды:

```
d8 k get mc deckhouse -o yaml
```

Пример вывода:

```
spec:
  settings:
    releaseChannel: Stable
    update:
      mode: AutoPatch
    windows:
      - days:
          - Mon
        from: "19:00"
        to: "20:00"
```

Как применяется автоматическое обновление

Автоматический режим активируется при указании параметра `releaseChannel` в конфигурации модуля `deckhouse`. При выполнении этого условия:

1. DKP раз в минуту проверяет канал обновлений на наличие нового релиза.
2. При появлении нового релиза DKP скачивает его в кластер и создает кастомный ресурс `DeckhouseRelease`.
3. После появления кастомного ресурса `DeckhouseRelease` в кластере DKP выполняет обновление на соответствующую версию согласно установленным параметрам обновления (по умолчанию — автоматически, в любое время).

Чтобы посмотреть список и состояние всех релизов в кластере, воспользуйтесь командой:

```
d8 k get deckhousereleases
```



Начиная с версии DKP 1.70, патч-версии обновлений (например, обновление на версию 1.70.2 при установленной версии 1.70.1) устанавливаются с учетом окон обновлений. До версии DKP 1.70 патч-версии обновлений устанавливаются без учета режима и окон обновлений.

Закрепление релиза

Под *закреплением релиза* подразумевается полное или частичное отключение автоматического обновления.

Есть три варианта ограничения автоматического обновления Deckhouse:

- Установить режим ручного подтверждения обновлений.

В этом случае DKP будет получать обновления в кластер, но для применения патч-версий и минорных версий потребуется ручное подтверждение.

Чтобы установить режим ручного подтверждения обновлений, задайте значение `Manual` для параметра `settings.update.mode` в настройках модуля `deckhouse` с помощью следующей команды:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
```

Чтобы подтвердить обновление, выполните следующую команду, указав необходимую версию DKP вместо `<DECKHOUSE-VERSION>`:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

- Установить режим автоматического обновления для патч-версий.

В этом случае DKP будет получать обновления в кластер, но для применения минорных версий потребуется ручное подтверждение. Патч-версии в рамках текущей минорной версии будут устанавливаться автоматически с учётом окон обновлений, если они заданы.

Например, если у вас установлена версия DKP `v1.70.1`, после установки этого режима Deckhouse сможет автоматически обновиться до версии `v1.70.2`, но не будет обновляться до версии `v1.71.*` без ручного подтверждения.

Чтобы установить режим автоматического обновления для патч-версий, задайте значение `AutoPatch` для параметра `settings.update.mode` в настройках модуля `deckhouse` с помощью следующей команды:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"AutoPatch"}}}}'
```

Чтобы подтвердить обновление для минорной версии, выполните следующую команду, указав необходимую версию DKP вместо `<DECKHOUSE-VERSION>`:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

- Установить тег необходимой версии DKP для Deployment `deckhouse` и удалить параметр `releaseChannel` из конфигурации модуля `deckhouse`.

В этом случае DKP останется на указанной версии и никакая информация о новых доступных версиях (объекты `DeckhouseRelease`) в кластере появляться не будет.

i Важно. Этот режим заблокирует установку патч-релизов, которые могут содержать исправления критических уязвимостей и ошибок. Если нужно получать патчи в рамках текущей минорной версии, используйте режим `AutoPatch` вместо жёсткого закрепления.

Пример установки версии `v1.66.3` для DKP EE и удаления параметра `releaseChannel` из конфигурации модуля `deckhouse`:

```
d8 k -ti -n d8-system exec svc/deckhouse-leader -C deckhouse -- kubectl set image deployment/deckhouse deckhouse=registry.deckhouse.ru/deckhouse/ee:v1.66.3
```

```
d8 k patch mc deckhouse --type=json -p='[{"op": "remove", "path": "/spec/settings/releaseChannel"}]'
```

Ручное подтверждение обновлений

Ручное подтверждение обновления версии DKP предусмотрено в следующих случаях:

- Включен режим подтверждения обновлений DKP.

Это значит, что для параметра `settings.update.mode` модуля `deckhouse` задано значение `Manual` (подтверждение как патч-версии, так и минорной версии DKP) или `AutoPatch` (подтверждение минорной версии DKP).

Для подтверждения обновления выполните следующую команду, указав необходимую версию DKP вместо `<DECKHOUSE-VERSION>`:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

- Если для какой-либо группы узлов отключено автоматическое применение обновлений, которые могут привести к кратковременному простоя в работе системных компонентов.

Это значит, что для параметра `spec.disruptions.approvalMode` у соответствующего ресурса `NodeGroup` задано значение `Manual`.

Для обновления узлов в такой группе установите аннотацию `update.node.deckhouse.io/disruption-approved=` на каждый узел.

Пример:

```
d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
```

Окна обновлений

DKP позволяет задавать *окна обновлений* — временные интервалы, в которые будет выполняться установка обновлений в автоматическом режиме. Используя окна обновлений, вы исключаете вероятность установки нового релиза в неподходящее время или в периоды высокой нагрузки на кластер.

Установка обновлений при настроенных окнах обновлений

- Если окна обновлений настроены, DKP будет устанавливать новые версии только в указанные временные интервалы.
- Если окна обновлений не настроены, установка начнется сразу после появления новой версии в настроенном канале обновлений.

Настройка окон обновлений

Управлять окнами обновлений DKP можно следующими способами:

- **для общего управления обновлениями** используйте параметр `update.windows` модуля `deckhouse`;
- **для управления обновлениями, которые могут привести к кратковременному простоя в работе системных компонентов**, используйте параметры `disruptions.automatic.windows` и `disruptions.rollingUpdate.windows` ресурса `NodeGroup`.

Примеры конфигурации

- Два ежедневных окна обновлений с 8:00 до 10:00 и с 20:00 до 22:00 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: EarlyAccess
    update:
      windows:
        - from: "8:00"
          to: "10:00"
        - from: "20:00"
          to: "22:00"
```

- Окна обновлений по вторникам и субботам с 18:00 до 19:30 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
      days:
        - Tue
        - Sat
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `deckhouse`
- Кастомные ресурсы модуля `node-manager`

Настройка уведомлений о новых релизах

Deckhouse Kubernetes Platform (DKP) генерирует алерты в систему мониторинга и может автоматически отправлять уведомления о будущих минорных обновлениях во внешние системы. Это помогает заранее планировать обновления и подготовиться к ним.

Условия отправки уведомлений во внешние системы:

- для DKP установлен автоматический режим обновления;
- планируется смена минорной версии (уведомления о патч-версиях не отправляются);
- настроен вебхук для уведомлений.

Алерты в системе мониторинга

Если обновление требует внесения изменений в кластер (например, обновление версии Kubernetes или ОС), DKP сгенерирует специальные предупреждения. Среди них:

- `D8NodeHasDeprecatedOSVersion` — в кластере обнаружены узлы с неподдерживаемой версией ОС;
- `HelmReleasesHasResourcesWithDeprecatedVersions` — в некоторых Helm-релизах используются устаревшие ресурсы;
- `D8KubernetesVersionIsDeprecated` — установленная версия Kubernetes более не поддерживается.

При появлении данных предупреждений обязательно устраните их причины перед обновлением. Это поможет избежать сбоев в работе кластера и обеспечить его стабильность после обновления.

Настройка уведомлений

В режиме обновлений `Auto` можно настроить вызов вебхука для получения оповещения о предстоящем обновлении минорной версии DKP.

Кроме того, оповещения формируются не только при обновлении DKP, но и при обновлении любых модулей, включая их отдельные обновления. В отдельных случаях система может инициировать отправку нескольких оповещений одновременно (по 10-20 оповещений) с интервалом около 15 секунд.

i Оповещения доступны только в режиме обновлений `Auto`, в режиме `Manual` они не формируются.

i Вебхук указывать не обязательно: если параметр `update.notification.webhook` не задан, но указано время в параметре `update.notification.minimalNotificationTime`, применение новой версии всё равно будет отложено на указанный период. В этом случае оповещением о появлении новой версии можно считать появление в кластере ресурса `DeckhouseRelease` с именем новой версии.

После появления новой минорной версии DKP на используемом канале обновлений, но до момента её применения в кластере, на указанный адрес вебхука будет выполнен POST-запрос.

Параметр `minimalNotificationTime` позволяет отложить установку обновления на заданный период, обеспечивая время для реакции на оповещение с учётом окон обновлений. Если при этом вебхук недоступен, каждая неудачная попытка отправки будет сдвигать время применения на ту же величину, что может привести к бесконечному откладыванию обновления.



Если ваш вебхук возвращает любой код вне диапазона 2xx, DKP повторяет отправку уведомления до пяти раз с экспоненциальной задержкой между попытками. Если все попытки окажутся неуспешными, выпуск блокируется до восстановления доступности вебхука.

Для удобной обработки ошибок и отладки при возврате кодов ошибок вебхук должен возвращать JSON-ответ следующей структуры:

- `code` — необязательный внутренний код ошибки для программной обработки;
- `message` — человекочитаемое описание того, что пошло не так.

Если вебхук возвращает успешный статус HTTP (2xx), DKP считает уведомление успешным вне зависимости от содержимого ответа.

Минимальный пример вебхука на Go...

```
package main
```

```

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
)
// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
    Subject      string          `json:"subject"`
    ModuleName   string          `json:"moduleName,omitempty"`
    Version      string          `json:"version"`
    Requirements map[string]string `json:"requirements,omitempty"`
    ChangelogLink string         `json:"changelogLink,omitempty"`
    ApplyTime    string          `json:"applyTime,omitempty"`
    Message      string          `json:"message"`
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
    Code    string `json:"code,omitempty"`
    Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
    }
    defer r.Body.Close()

    var data WebhookData
    if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
        log.Printf("failed to decode payload: %v", err)
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    // Print payload fields
    log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
        data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
    log.Printf("message=%s", data.Message)

    // Example conditional logic: fail intentionally for testing
    if data.Version == "v0.0.0-fail" {
        // Return structured error response with error status code
        errorResp := ResponseError{
            Code:    "TEST_FAILURE",
            Message: "intentional failure for testing",
        }

        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode(errorResp)
        return
    }

    // Return success response with 2xx status code
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Notification processed successfully"))
}

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/webhook", handler)

    addr := ":8080"
    fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
    if err := http.ListenAndServe(addr, mux); err != nil {

```

```

    log.Fatal(err)
  }
}

```

Поддерживаемые параметры

- `update.notification.webhook` — URL-адрес для отправки уведомлений. POST-запрос с информацией об обновлении отправляется сразу после появления новой минорной версии на используемом канале обновлений, но до установки обновления в кластере.
- `update.notification.auth` — параметры аутентификации при вызове вебхука. Если параметр не указан, аутентификация не используется.
 - `update.notification.auth.basic` — базовый вариант аутентификации. Имя пользователя и пароль передаются в заголовке `Authorization` в формате `Basic <base64(username:password)>`.
 - `update.notification.auth.basic.username` — имя пользователя.
 - `update.notification.auth.basic.password` — пароль.
 - `update.notification.auth.bearerToken` — аутентификация с использованием токена. Токен передаётся в заголовке `Authorization` в формате `Bearer <token>`.
- `update.notification.minimalNotificationTime` — минимальный интервал между появлением новой минорной версии на используемом канале обновлений и началом обновления. Указывается в часах и минутах: `30m`, `1h`, `2h30m`, `24h`. Если настроено окно обновлений, то сначала учитывается интервал `minimalNotificationTime`, после чего произойдёт установка обновления, но только в рамках заданного окна.
- `update.notification.tlsSkipVerify` — отключение проверки TLS-сертификата при вызове вебхука (например, если используется самоподписанный сертификат). По умолчанию `false`.

Пример настройки `update.notification` с базовой аутентификацией:

```

update:
  notification:
    webhook: https://release-webhook.mydomain.com
    minimalNotificationTime: 4h
  auth:
    basic:
      username: myusername
      password: mypassword
  tlsSkipVerify: true

```

Пример настройки `update.notification` без аутентификации:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h

```

Формат уведомлений

При выполнении условий отправки уведомления DKP отправляет на указанный вебхук POST-запрос с заголовком `Content-Type: application/json`.

Пример тела запроса для релиза Deckhouse:

```
{
  "subject": "Deckhouse",
  "version": "1.68",
  "requirements": { "k8s": "1.29.0" },
  "changelogLink":
    "https://github.com/deckhouse/deckhouse/blob/main/CHANGELOG/CHANGELOG-v1.68.md",
  "applyTime": "2025-02-01T14:30:00Z00:00",
  "message": "New Deckhouse Release 1.68 is available. Release will be applied at:
    Wednesday, 05-Feb-25 14:30:00 UTC"
}
```

Пример тела запроса для релиза модуля:

```
{
  "subject": "Module",
  "moduleName": "test",
  "version": "0.7.14",
  "requirements": { "deckhouse": ">= 1.69" },
  "changelogLink": "https://github.com/deckhouse/modules/blob/main/CHANGELOG/CHANGELOG-
    test-v0.7.14.md",
  "applyTime": "2025-12-17T12:32:24Z",
  "message": "New test Release 0.7.14 is available. Release will be applied at:
    Wednesday, 17-Dec-25 12:32:24 UTC"
}
```

Описание полей в формате уведомлений:

- `subject` — субъект релиза (“Deckhouse” для релизов Deckhouse или “Module” для релизов модулей);
- `moduleName` — имя модуля (присутствует только для релизов модулей);
- `version` — номер минорной версии (строка);
- `requirements` — объект с требованиями к новой версии (например, минимальная версия Kubernetes);
- `changelogLink` — ссылка на changelog с описанием изменений в новой минорной версии;
- `applyTime` — дата и время планируемого обновления в формате RFC3339. Учитывает заданные окна обновлений;
- `message` — короткое текстовое уведомление о доступной минорной версии и запланированном времени её установки.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `deckhouse`

Обновление Kubernetes и управление версиями

Обновление и управление версиями

Процесс обновления control plane в DKP полностью автоматизирован.

- В DKP поддерживаются последние пять версий Kubernetes.
- Control plane можно откатывать на одну минорную версию назад и обновлять на несколько версий вперёд — шаг за шагом, по одной версии за раз.
- Patch-версии (например, `1.27.3` → `1.27.5`) обновляются автоматически вместе с версией Deckhouse, и управлять этим процессом нельзя.
- Minor-версии задаются вручную в параметре `kubernetesVersion` в ресурсе `ClusterConfiguration`.

Изменение версии Kubernetes

Понижение версии возможно только на одну минорную назад от максимальной версии, когда-либо использовавшейся в кластере. Обновление вперёд — по одной минорной версии за раз.

1. Откройте редактирование `ClusterConfiguration`:

```
d8 system edit cluster-configuration
```

2. Установите желаемую версию Kubernetes (`kubernetesVersion`):

```

apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
cloud:
  prefix: demo-stand
  provider: Yandex
clusterDomain: cloud.education
clusterType: Cloud
defaultCRI: Containerd
kubernetesVersion: "1.30"
podSubnetCIDR: 10.111.0.0/16
podSubnetNodeCIDRPrefix: "24"
serviceSubnetCIDR: 10.222.0.0/16

```

3. Сохраните изменения.
4. Дождитесь окончания обновления. Отслеживать ход обновления можно с помощью команды `d8 k get po`. Обновление можно считать завершённым, когда в выводе команды у каждого узла кластера в колонке `VERSION` появится обновлённая версия.

Мониторинг процесса обновления Kubernetes

Модуль `control-plane-manager` включает компонент `update-observer`, который предоставляет актуальную информацию о процессе обновления Kubernetes в кластере.

Компонент `update-observer`:

- Читает конфигурацию кластера из секрета `d8-cluster-configuration`;
- Отслеживает версии kubelet на всех узлах через `nodeInfo.kubeletVersion`;
- Собирает версии всех экземпляров control plane по аннотации `control-plane-manager.deckhouse.io/kubernetes-version`;
- Создаёт и обновляет ConfigMap `d8-cluster-kubernetes` в пространстве имён `kube-system` с подробным статусом обновления.

В ConfigMap `d8-cluster-kubernetes` отображаются:

- **Статус компонентов** — версии компонентов control plane (`kube-apiserver`, `kube-scheduler`, `kube-controller-manager`) на каждом master-узле;
- **Прогресс обновления узлов** — сколько узлов уже обновлено и сколько всего узлов должно быть обновлено;
- **Целевая и текущая версии** — желаемая версия из конфигурации и фактическое состояние кластера во время обновления;
- **Расхождение версий** — информация о компонентах, работающих не на целевой версии (в том числе на версии выше целевой);
- **Списки версий**:
 - `supportedVersions` — минорные версии Kubernetes, поддерживаемые в текущем релизе DKP;
 - `availableVersions` — версии, доступные для выбора при обновлении или понижении в данном кластере. Набор ограничен максимальной когда-либо установленной минорной версией и правилом одного минорного шага при откате;
 - `automaticVersion` — минорная версия, которая будет использована при режиме обновления `Automatic`.

В фазе `ControlPlaneUpdating` поле `status.progress` отражает общий прогресс обновления с учётом промежуточных минорных версий. При многошаговом обновлении (например, `1.33 → 1.35`) процент растёт по мере завершения каждого шага, а не только когда все компоненты control plane достигнут финальной целевой версии.

Минорные версии в ConfigMap (`spec`, `status`, а также метки `k8s-version` и `max-k8s-version`) задаются в том же формате, что и в `ClusterConfiguration` — без префикса `v` (например, `"1.33"`).

Это позволяет в реальном времени видеть, какие компоненты обновляются, на каком этапе находится процесс, и не остановилось ли обновление на каком-либо узле или компоненте.

Для просмотра статуса обновления выполните команду:

```
d8 k get configmap d8-cluster-kubernetes -n kube-system -o yaml
```

Примеры содержимого ConfigMap

В `data.spec` и `data.status` хранится YAML с полем `spec` (целевая версия и режим обновления) и полем `status` (текущее состояние). Ниже приведены примеры для разных сценариев.

Кластер в актуальном состоянии (3 master-узла, 3 worker-узла)

```

apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.32"
    supportedVersions:
      - "1.30"
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    availableVersions:
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    automaticVersion: "1.33"
    phase: UpToDate
    controlPlane:
      - name: master-1
        phase: UpToDate
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
      - name: master-2
        phase: UpToDate
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
      - name: master-0
        phase: UpToDate
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: idle
    lastReconciliationTime: "2026-02-02T01:13:05Z"
    lastUpToDateTime: "2026-01-30T16:26:36Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.32"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "20837731"
  uid: ba981996-f737-469c-9ce1-53aa46135994

```

Начало обновления (например, понижение версии Kubernetes)

Целевая версия уже задана, control plane или узлы ещё переходят на неё.

```

apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
      - "1.30"
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    availableVersions:
      - "1.32"
      - "1.33"
      - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 0%
    controlPlane:
      - name: master-0
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-1
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-2
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:34:42Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21379847"
  uid: ba981996-f737-469c-9ce1-53aa46135994

```

Обновление control plane в процессе

Часть master-узлов уже на новой версии, часть ещё обновляется. В `status.progress` отображается прогресс в процентах.

```

apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
      - "1.30"
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    availableVersions:
      - "1.32"
      - "1.33"
      - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
      - name: master-0
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-1
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-2
        phase: UpToDate
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:41:55Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21388343"
  uid: ba981996-f737-469c-9ce1-53aa46135994

```

Промежуточный шаг многошагового обновления (например, 1.33 → 1.35)

В конфигурации целевая версия Kubernetes может опережать текущую версию кластера на несколько минорных версий. В этом случае обновление выполняется поэтапно, через промежуточные минорные версии.

Поле `status.currentVersion` отражает текущий шаг обновления (активную минорную версию, до которой в данный момент доводится кластер). При этом в рамках одного шага отдельные компоненты control plane могут временно работать на разных минорных версиях.

Поле `progress` учитывает весь путь обновления, включая промежуточные версии. По этой причине его значение может быть существенно больше 0 % ещё до того, как все компоненты достигнут целевой версии.

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.35"
    updateMode: Manual
  status: |
    currentVersion: "1.34"
    supportedVersions:
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
      - "1.35"
    availableVersions:
      - "1.33"
      - "1.34"
      - "1.35"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
      - name: master-0
        phase: Updating
        components:
          kube-apiserver: "1.35"
          kube-controller-manager: "1.34"
          kube-scheduler: "1.34"
      - name: master-1
        phase: Updating
        components:
          kube-apiserver: "1.34"
          kube-controller-manager: "1.34"
          kube-scheduler: "1.34"
      - name: master-2
        phase: Updating
        components:
          kube-apiserver: "1.34"
          kube-controller-manager: "1.34"
          kube-scheduler: "1.34"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-04-17T09:43:08Z"
    lastUpToDateTime: "2026-04-15T13:43:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "1753458"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Кластер в актуальном состоянии (2 master-узла, 1 arbitr-узел и 3 worker-узла)

```

apiVersion: v1
data:
  spec: |
    desiredVersion: "1.33"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
      - "1.30"
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    availableVersions:
      - "1.32"
      - "1.33"
      - "1.34"
    automaticVersion: "1.33"
    phase: UpToDate
    controlPlane:
      - name: master-0
        phase: UpToDate
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-1
        phase: UpToDate
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-02-02T11:09:59Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21357074"
  uid: ba981996-f737-469c-9ce1-53aa46135994

```

Сбой одного или нескольких компонентов control plane

У master-узла `phase: Failed`, в поле `description` — причина (например, под или контейнер не в состоянии `Running`):

```

apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.32"
    supportedVersions:
      - "1.30"
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    availableVersions:
      - "1.31"
      - "1.32"
      - "1.33"
      - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 73%
    controlPlane:
      - name: master-1
        phase: UpToDate
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
      - name: master-2
        phase: Updating
        components:
          kube-apiserver: "1.33"
          kube-controller-manager: "1.33"
          kube-scheduler: "1.33"
      - name: master-0
        phase: Failed
        components:
          kube-apiserver: "1.32"
          kube-controller-manager: "1.32"
          kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:

```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `control-plane-manager`

Удаление

Удаление кластера, развёрнутого в облачном провайдере

Для удаления кластера, развёрнутого в облачном провайдере, нужно выполнить несколько шагов:

1. Узнайте канал обновления, заданный в кластере. Для этого выполните команду:

```
kubectl get mc deckhouse -o jsonpath='{.spec.settings.releaseChannel}'
```

2. Запустите инсталлятор Deckhouse:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
registry.deckhouse.ru/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

где:

- **<MOUNT_OPTIONS>** — параметры монтирования файлов в контейнер инсталлятора, таких как SSH-ключи доступа;
- **<DECKHOUSE_REVISION>** — редакция Deckhouse (например, **ee** — для Enterprise Edition, **ce** — для Community Edition и т. д.)
- **<RELEASE_CHANNEL>** — канал обновлений Deckhouse в kebab-case:
 - **alpha** — для канала обновлений *Alpha*;
 - **beta** — для канала обновлений *Beta*;
 - **early-access** — для канала обновлений *Early Access*;
 - **stable** — для канала обновлений *Stable*;
 - **rock-solid** — для канала обновлений *Rock Solid*.

Пример запуска контейнера инсталлятора Deckhouse CE:

```
docker run -it --pull=always \
-v "$PWD/dhctl-tmp:/tmp/dhctl" \
-v "$HOME/.ssh:/tmp/.ssh/" registry.deckhouse.ru/deckhouse/ce/install:stable bash
```

3. В запущившемся контейнере выполните команду:

```
dhctl destroy --ssh-user=<USER> \
--ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
--yes-i-am-sane-and-i-understand-what-i-am-doing \
--ssh-host=<MASTER_IP>
```

где:

- **<USER>** — пользователь удалённой машины, из-под которого производилась установка. Если для **<USER>** требуется выполнять действия через **sudo**, добавьте флаг **--ask-become-pass**;
- **<MASTER_IP>** — IP-адрес master-узла кластера;
- **<SSH_PRIVATE_KEY_FILE>** — имя приватного ключа. Например, для ключа с RSA-шифрованием это может быть **id_rsa**, а для ключа с ED25519-шифрованием — **id_ed25519**.

Инсталлятор подключится к кластеру, получит необходимые данные и произведёт удаление всех ресурсов и объектов в облаке, созданных при установке и работе DKP.

Удаление гибридного кластера

Для удаления гибридного кластера, состоящего из автоматически развёрнутых узлов в облаке и вручную подключённых статических узлов, выполните следующие действия:

1. Удалите из кластера все вручную подключённые дополнительные узлы — CloudStatic и Static.
2. Узнайте канал обновления, заданный в кластере. Для этого выполните команду:

```
kubectl get mc deckhouse -o jsonpath='{.spec.settings.releaseChannel}'
```

3. Запустите инсталлятор Deckhouse:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
registry.deckhouse.ru/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

где:

- **<MOUNT_OPTIONS>** — параметры монтирования файлов в контейнер инсталлятора, таких как SSH-ключи доступа;
- **<DECKHOUSE_REVISION>** — редакция Deckhouse (например, **ee** — для Enterprise Edition, **ce** — для Community Edition и т. д.)
- **<RELEASE_CHANNEL>** — канал обновлений Deckhouse в kebab-case:
 - **alpha** — для канала обновлений *Alpha*;
 - **beta** — для канала обновлений *Beta*;
 - **early-access** — для канала обновлений *Early Access*;
 - **stable** — для канала обновлений *Stable*;
 - **rock-solid** — для канала обновлений *Rock Solid*.

Пример запуска контейнера инсталлятора Deckhouse CE:

```
docker run -it --pull=always \
-v "$PWD/dhctl-tmp:/tmp/dhctl" \
-v "$HOME/.ssh:/tmp/.ssh/" registry.deckhouse.ru/deckhouse/ce/install:stable bash
```

4. В запущившемся контейнере выполните команду:

```
dhctl destroy --ssh-user=<USER> \
--ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
--yes-i-am-sane-and-i-understand-what-i-am-doing \
--ssh-host=<MASTER_IP>
```

где:

- **<USER>** — пользователь удалённой машины, из-под которого производилась установка. Если для **<USER>** требуется выполнять действия через **sudo**, добавьте флаг **--ask-become-pass**;
- **<MASTER_IP>** — IP-адрес master-узла кластера;
- **<SSH_PRIVATE_KEY_FILE>** — имя приватного ключа. Например, для ключа с RSA-шифрованием это может быть **id_rsa**, а для ключа с ED25519-шифрованием — **id_ed25519**.

Инсталлятор подключится к кластеру, получит необходимые данные и произведёт удаление всех ресурсов и объектов в облаке, созданных при установке и работе DKP.

Удаление статического кластера

Для удаления кластера, установленного вручную (например, bare metal), нужно выполнить несколько шагов:

1. Удалите из кластера все дополнительные узлы.
2. Узнайте канал обновления, заданный в кластере. Для этого выполните команду:

```
kubectrl get mc deckhouse -o jsonpath='{.spec.settings.releaseChannel}'
```

3. Запустите инсталлятор Deckhouse:

```
docker run --pull=always -it [<MOUNT_OPTIONS>] \
registry.deckhouse.ru/deckhouse/<DECKHOUSE_REVISION>/install:<RELEASE_CHANNEL> bash
```

где:

- **<MOUNT_OPTIONS>** — параметры монтирования файлов в контейнер инсталлятора, таких как SSH-ключи доступа;
- **<DECKHOUSE_REVISION>** — редакция Deckhouse (например, **ee** — для Enterprise Edition, **ce** — для Community Edition и т. д.)
- **<RELEASE_CHANNEL>** — канал обновлений Deckhouse в kebab-case:
 - **alpha** — для канала обновлений *Alpha*;
 - **beta** — для канала обновлений *Beta*;
 - **early-access** — для канала обновлений *Early Access*;
 - **stable** — для канала обновлений *Stable*;
 - **rock-solid** — для канала обновлений *Rock Solid*.

Пример запуска контейнера инсталлятора Deckhouse CE:

```
docker run -it --pull=always \
-v "$PWD/dhctl-tmp:/tmp/dhctl" \
-v "$HOME/.ssh:/tmp/.ssh/" registry.deckhouse.ru/deckhouse/ce/install:stable bash
```

4. Выполните команду удаления кластера:

```
dhctl destroy --ssh-user=<USER> \
--ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
--yes-i-am-sane-and-i-understand-what-i-am-doing \
--ssh-host=<MASTER_IP>
```

где:

- `<USER>` — пользователь удалённой машины, из-под которого производилась установка. Если для `<USER>` требуется выполнять действия через `sudo`, добавьте флаг `--ask-become-pass`;
- `<MASTER_IP>` — IP-адрес master-узла кластера;
- `<SSH_PRIVATE_KEY_FILE>` — имя приватного ключа. Например, для ключа с RSA-шифрованием это может быть `id_rsa`, а для ключа с ED25519-шифрованием — `id_ed25519`.

Инсталлятор подключится к master-узлу и удалит на нём все компоненты Deckhouse и кластера Kubernetes.

Дополнительные ресурсы

- Параметры модуля `deckhouse`
- Документация модуля `node-manager`

Архитектура

В данном разделе документации описана архитектура Deckhouse Kubernetes Platform (DKP).

Раздел состоит из следующих подразделов:

- Модель C4 — обзор модели C4, используемой для визуализации архитектуры платформы, а также описание архитектуры DKP на уровнях 1 и 2 модели C4.
- Модули — описание архитектуры модулей DKP.
- Катастрофоустойчивость — описание реализованных в DKP подходов к обеспечению катастрофоустойчивости.
- Обновление — описание механизмов обновления DKP.
- Описание архитектуры компонентов платформы, сгруппированных по следующим подсистемам:
 - Подсистема Deckhouse
 - Подсистема Kubernetes & Scheduling
 - Подсистема Cluster & Infrastructure
 - Подсистема IAM
 - Подсистема Security
 - Подсистема Network
 - Подсистема Storage
 - Подсистема Observability

i

В разделе представлена информация не по всем подсистемам и модулям DKP. Материалы по остальным компонентам будут добавляться по мере готовности.

Архитектура DKP

DKP — это платформа для управления кластерами Kubernetes в любых инфраструктурах — от изолированных серверных сред до публичных облаков. Платформа включает в себя:

- кластер Kubernetes;
- контроллер Deckhouse и управляемые им модули;
- Bashible — агент, работающий на узлах кластера в виде службы, который запускает bash-скрипты для управления узлами.

Модули объединены в подсистемы в соответствии с их функциональным назначением. Контроллер Deckhouse тоже реализован в виде модуля и является единственным модулем, без которого не может функционировать платформа.

Архитектура DKP в масштабе подсистем и модулей описана в подразделе Модель C4.

Модули

Модуль — это набор ресурсов и приложений, предназначенных для расширения функциональности DKP.

Ключевые модули:

- `deckhouse` — контроллер Deckhouse;
- `control-plane-manager` — управляет компонентами control plane кластера;
- `node-manager` — управляет узлами кластера.

i

При установке DKP в существующий Managed Kubernetes-кластер модули `control-plane-manager` и `node-manager` не устанавливаются.

Содержимое модулей:

- Helm-чарты;
- хуки `addon-operator`;
- правила сборки компонентов модуля (компонентов Deckhouse);
- другие файлы.

При работе с модулями, DKP использует проект `addon-operator`. Ознакомьтесь с его документацией, чтобы узнать, как DKP работает с модулями, хуками модулей и параметрами модулей.

Об архитектуре модуля и разработке собственных модулей читайте в разделе Модули.

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Документация модуля `deckhouse`
- Документация модуля `node-manager`

Архитектура DKP (модель C4)

Для графического изображения архитектуры системы и объяснения декомпозиции системы на элементы используется Модель C4. Диаграммы в этой модели организуются по принципу увеличения масштаба и детализации элементов:

- **Уровень 1: диаграмма контекста.** Отражает положение системы во внешнем контексте: возможные пользовательские роли и другие системы, непосредственно участвующие в работе.
- **Уровень 2: диаграмма контейнеров.** Разбивает систему на контейнеры (основные подсистемы) и отражает основные пути взаимодействия между ними. Контейнер в модели C4 (не путать с Docker-контейнерами) — это исполняемая и развертываемая подсистема, например, исполняемый файл.
- **Уровень 3: диаграмма компонентов.** Разбивает каждый контейнер на компоненты и отражает взаимодействие компонентов между собой (внутри одного контейнера) и с другими контейнерами. Компонент в модели C4 — это программный модуль (подсистема) в коде контейнера (исполняемого файла). Этот уровень описан в архитектурных диаграммах каждого сервиса.
- **Уровень 4: диаграмма кода.** Описывает элементы кода, с помощью которых реализуются компоненты.

Из-за большого количества элементов на уровне 2 полная диаграмма контейнеров здесь не приводится. Вместо этого после диаграммы уровня 1 представлены следующие промежуточные диаграммы:

- диаграмма подсистем;
- диаграмма модулей.

Диаграммы уровня 2 и выше приведены в разделах, описывающих архитектуру отдельных модулей.

Чтобы получить общее представление об архитектуре Deckhouse Kubernetes Platform (DKP), достаточно изучить диаграмму уровня 1, а также диаграммы подсистем и модулей. Для детального изучения устройства каждого сервиса обратитесь к диаграммам уровня 2 и выше, представленным в соответствующих разделах.

Условные обозначения

Для элементов диаграмм используются следующие графические символы:

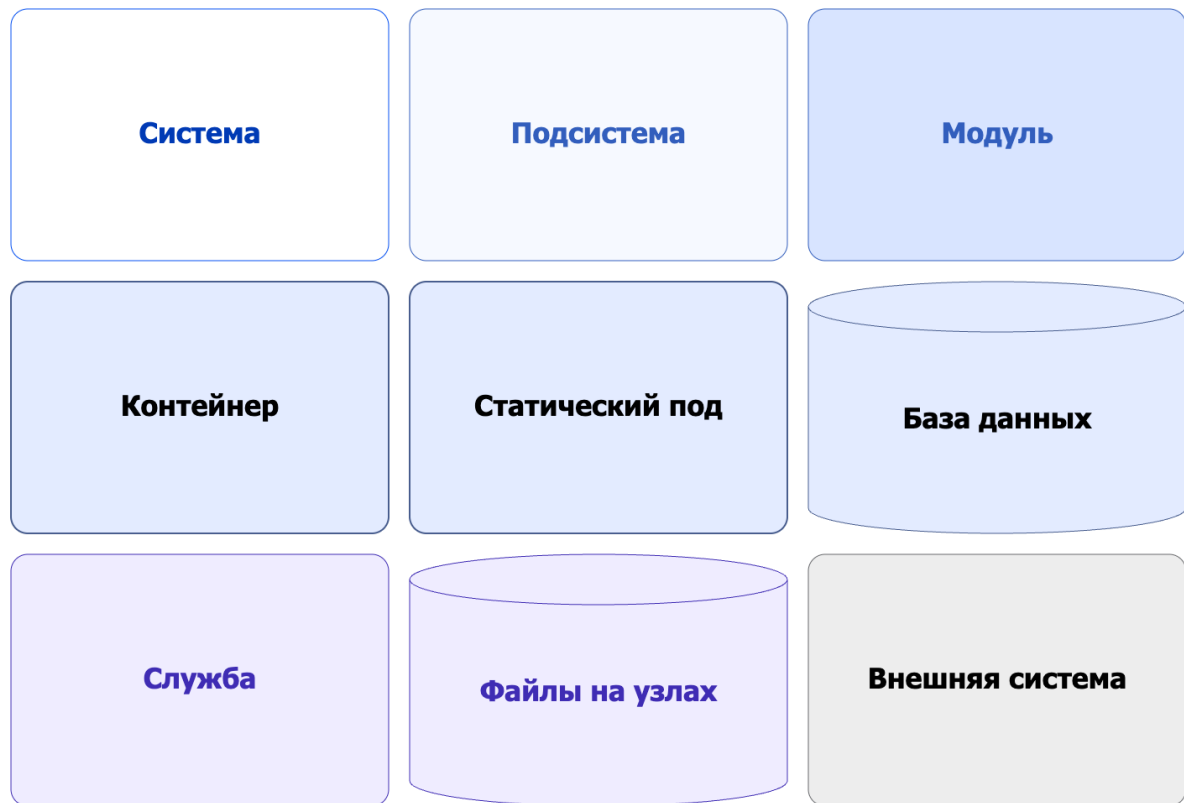
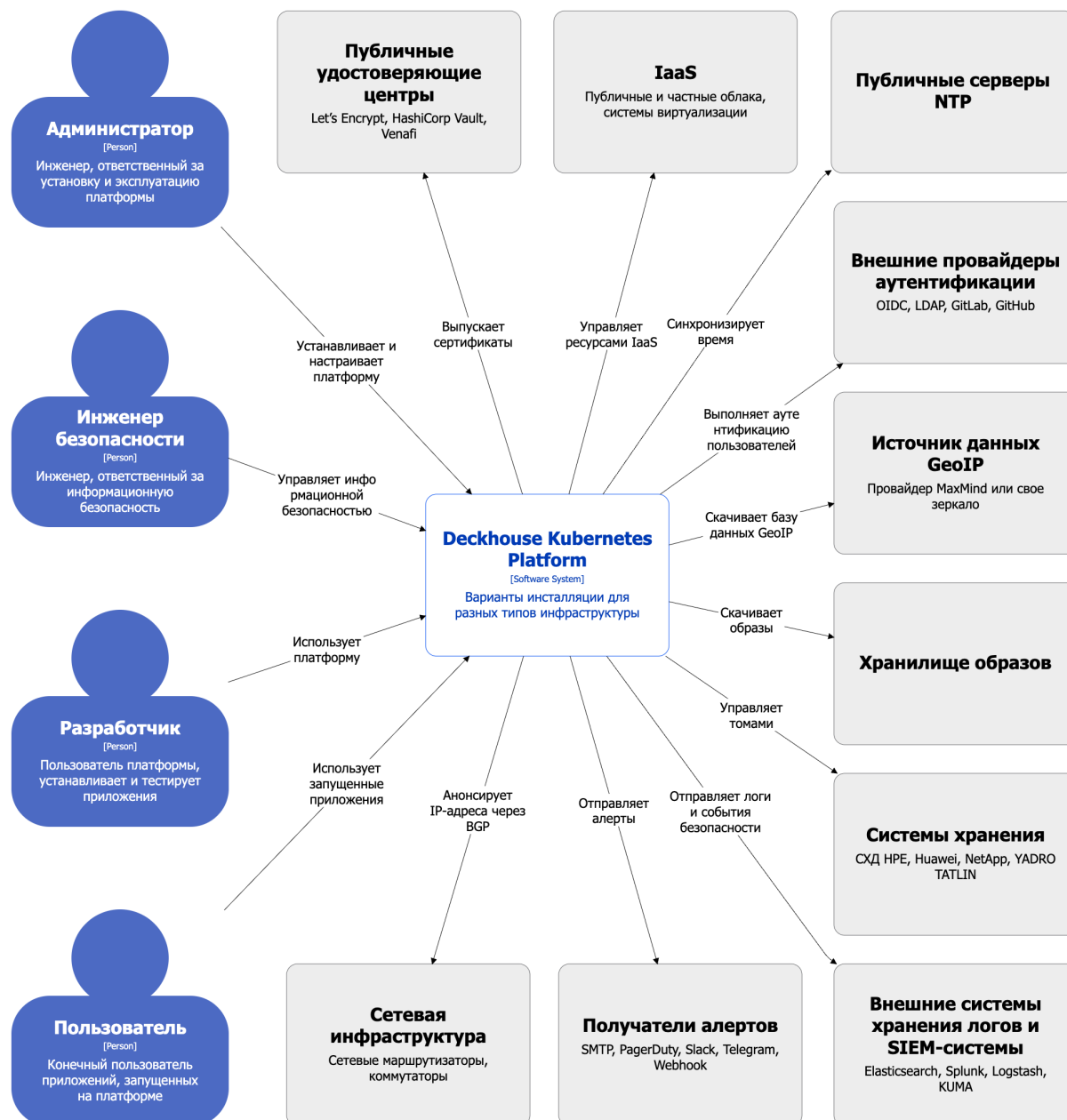


Диаграмма контекста



С DKP взаимодействуют:

- **Администратор** — устанавливает и настраивает платформу.
- **Инженер безопасности** — управляет информационной безопасностью.
- **Разработчик** — использует платформу для установки и тестирования разрабатываемых приложений.
- **Пользователь** — использует запущенные на платформе приложения.

Взаимодействие с внешними сервисами описано в диаграммах следующих уровней.

Диаграмма подсистем

На данной диаграмме изображены подсистемы DKP.



Связи между подсистемами показаны в обобщённом виде и без детализации.

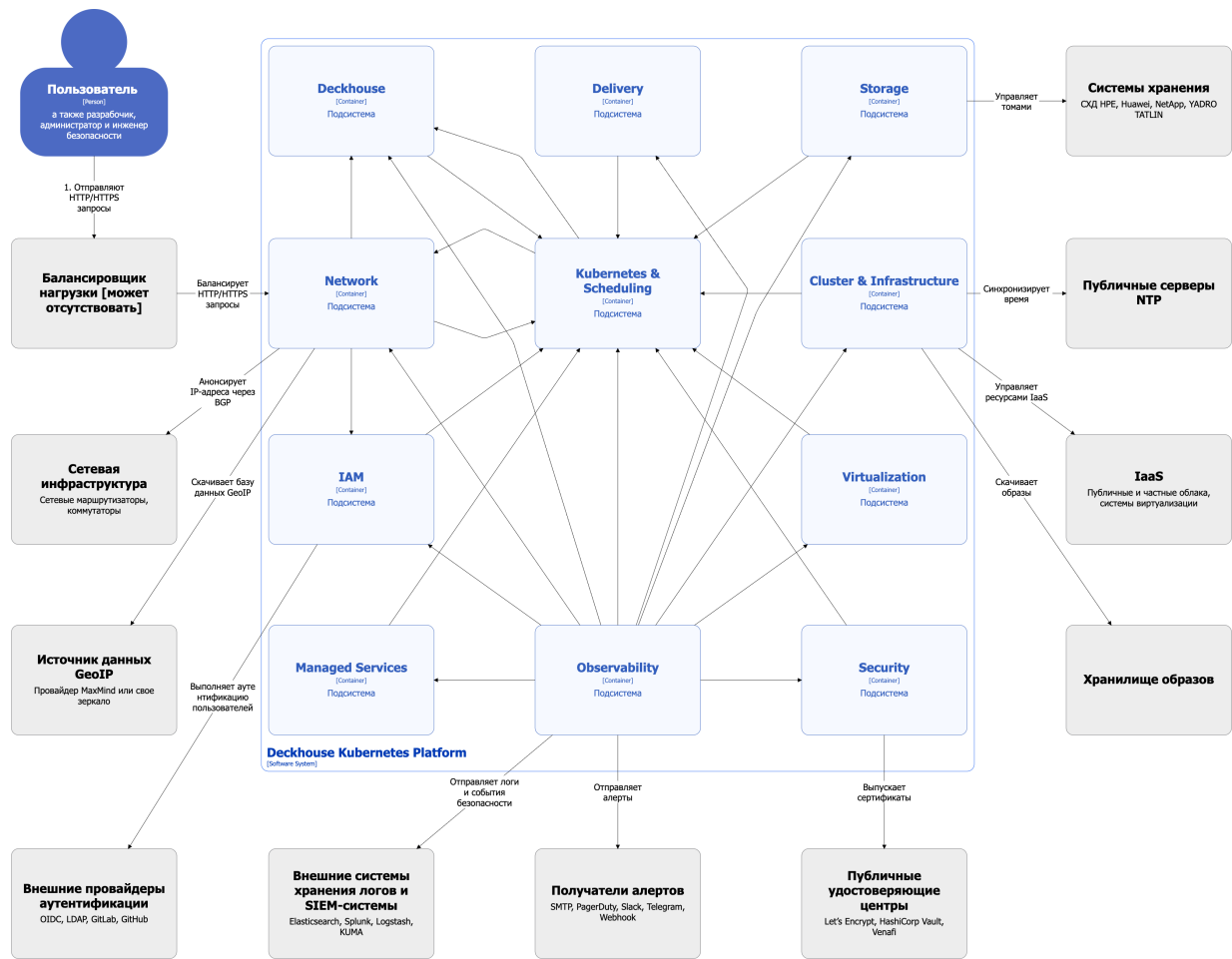
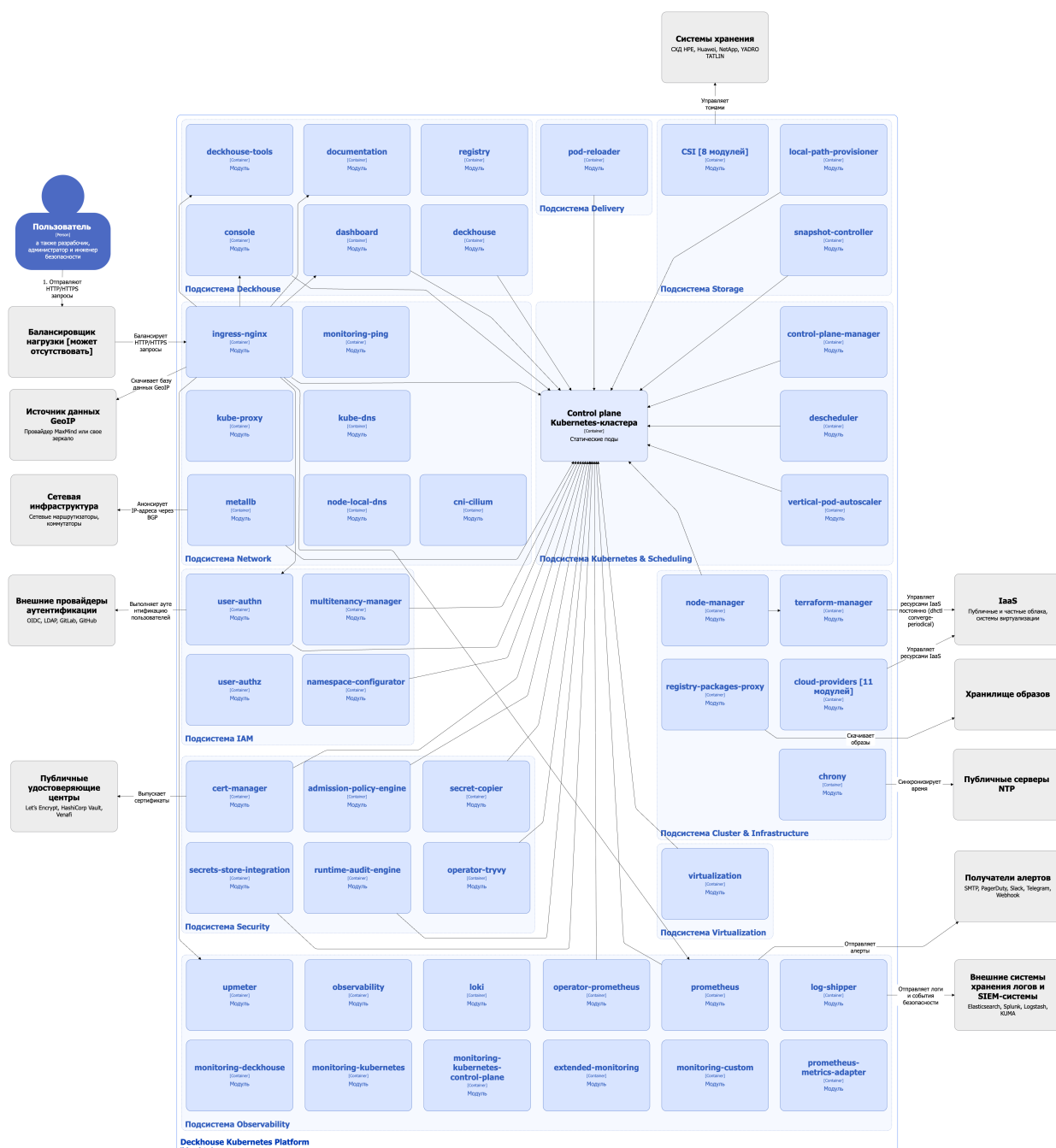


Диаграмма модулей

На данной диаграмме изображены все модули в составе набора модулей Default, а также наиболее важные дополнительные модули. Модули сгруппированы по подсистемам.

i Связи между модулями показаны в обобщённом виде и без детализации.



Дополнительные ресурсы

- The C4 model for visualising software architecture

Разработка модуля Deckhouse Kubernetes Platform

Модуль — набор ресурсов и приложений, предназначенных для расширения функциональности Deckhouse Kubernetes Platform.

Этот раздел посвящён архитектуре модулей DKP. Он поможет понять, как устроены модули, как их разрабатывать и отлаживать, а также как они взаимодействуют с другими компонентами платформы.

Deckhouse Kubernetes Platform (DKP) может работать со следующими видами модулей:

- Встроенные модули. Входят в состав DKP. Релизный цикл привязан к релизному циклу DKP.
- Модули из источника модулей. Релизный цикл таких модулей не привязан к релизному циклу DKP.

Создание модуля состоит из следующих этапов:

- Разработка**: создание кода модуля и его структуры хранения в Git-репозитории. В разделе **Структура модуля** описывается, какие компоненты существуют и в каких директориях находятся.
- Сборка**: создание артефакта модуля и загрузка его в хранилище образов контейнеров. В

разделе **Сборка и публикация** разбирается, какие образы хранятся в хранилище образов и по каким путям они доступны.

- **Запуск в кластере:** этап охватывает процесс доставки модуля в кластер на платформе DKP. В разделе **Запуск в кластере** описывается, как активировать модуль, настроить его параметры и проверить его работоспособность, включая работу с CRD и поиск возможных ошибок.
- **Зависимости:** этап настройки зависимостей модуля, включая версии DKP, Kubernetes и другие критичные компоненты, описывается в разделе **Зависимости модуля**.

Требования

Для разработки модулей DKP вам понадобятся следующие инструменты:

- git — система контроля версий;
- sed — редактор потоков;
- yq — CLI-утилита для обработки данных в форматах JSON, YAML и XML;
- jq — CLI-утилита для обработки данных в форматах JSON, YAML и XML;
- werf — (не обязательно) CLI-утилита для сборки образов. Она понадобится, если вы хотите собрать артефакты модуля локально;
- crane — (не обязательно) CLI-утилита для работы с хранилищами образов. Может понадобиться при отладке.

Хранилище образов, в котором содержатся артефакты модуля, должен поддерживать вложенную структуру репозитория. Можно использовать такие хранилища образов, как Docker Registry v2 или Harbor.

Прежде чем начать

Чтобы понять принципы, по которым работают модули DKP, ознакомьтесь с `addon-operator` и `shell-operator`.

- Прочитайте документацию операторов о концепции хуков, например, что такое конфигурация хука и какие функции она предоставляет. С помощью конфигурации настраиваются данные, которые будут доступны из хука.
- Ознакомьтесь с информацией о биндингах. Биндинги — это события, при которых срабатывает хук. Биндинги указываются в конфигурации хука. Хук может сработать не только из-за событий в Kubernetes, а также, например, по расписанию или перед запуском модуля.

i Хук позволяет сохранять значения в памяти и использовать их позже при рендеринге шаблонов Helm. Об этой особенности и цикле работы модуля рекомендуем прочитать в документации `Hooks and Helm values`.

- Ознакомьтесь с концепцией снапшотов. С помощью снапшотов можно реализовать подход `reconciliation loop`, более эффективный, чем подписка на события.

i Этим способом в DKP реализована поддержка всех существующих хуков внутренних модулей.

- Кроме того, хуки можно использовать вместо экспортера Prometheus. Хуки могут предоставлять метрики, которые DKP будет экспортировать. Ознакомьтесь с информацией о метриках.

Вопросы по документации

В разделе Разработка и отладка собрана информация о том, какие инструменты можно применять при разработке модуля, и о подходах к поиску ошибок в работе модуля.

Присоединяйтесь к сообществу, где вам обязательно помогут ответить на появившиеся вопросы.

Структура модуля

Исходный код модуля и правила его сборки должны находиться в директории с определённой структурой. Ближайший аналог — `Helm chart`.

Не все папки и файлы модуля обязательны. Кратко, можно руководствоваться следующим:

- Создайте файл `module.yaml`, в котором опишите метаданные модуля.
- В папке `templates` разместите шаблоны Helm, которые будут применяться в кластере.

Если необходимо чтобы объекты, создаваемые модулем, меняли своё поведение в зависимости от каких-либо параметров модуля, определите необходимые параметры в спецификации и используйте их в шаблонах.

- В папке `images` разместите инструкции по сборке образов контейнеров, используемых модулем.

Если в шаблонах используются только адреса *внешних* образов, папку можно не создавать.

- Если модуль должен реагировать на события или взаимодействовать с Kubernetes API, то необходимо создать хуки, которые разместить в папке `hooks`.
- Разместите документацию к модулю в папке `docs`. Если документация модуля отсутствует, то модуль не появится в списке модулей в веб-интерфейсе документации в кластере.
- Если у модуля есть вспомогательные чарты Helm, разместите их в папке `charts`.
- Если модуль должен создавать кастомные ресурсы (CRD), разместите их спецификации в папке `crds`.

Мы подготовили репозиторий шаблона модуля, содержащий структуру файлов и директорий, с которой мы рекомендуем начинать разработку модуля.

Далее на этой странице вы найдёте более подробное описание структуры директорий и файлов модуля.

▼ Пример структуры папки модуля...

Пример структуры папки модуля, содержащий правила сборки и публикации с помощью GitHub Actions:

```

├─ my-module/
│  ├─ .github/
│  │  └─ workflows/
│  │     ├─ build_dev.yaml
│  │     ├─ build_prod.yaml
│  │     ├─ checks.yaml
│  │     ├─ deploy_dev.yaml
│  │     └─ deploy_prod.yaml
│  └─ .werf/
│     └─ workflows/
│        ├─ base-images.yaml
│        ├─ bundle.yaml
│        ├─ images.yaml
│        ├─ images-digest.yaml
│        ├─ batch-go.yaml
│        └─ release.yaml
├─ charts/
│  └─ helm_lib/
├─ crds/
│  ├─ crd1.yaml
│  ├─ doc-ru-crd1.yaml
│  ├─ crd2.yaml
│  └─ doc-ru-crd2.yaml
├─ docs/
│  ├─ README.md
│  ├─ README.ru.md
│  ├─ EXAMPLES.md
│  ├─ EXAMPLES.ru.md
│  ├─ CONFIGURATION.md
│  ├─ CONFIGURATION.ru.md
│  ├─ CR.md
│  ├─ CR.ru.md
│  ├─ FAQ.md
│  ├─ FAQ.ru.md
│  ├─ ADVANCED_USAGE.md
│  └─ ADVANCED_USAGE.ru.md
├─ hooks/
│  └─ batch/
│     └─ my-hooks/
│        ├─ my-hook1.go
│        └─ my-hook2.go
│     └─ go.mod
│     └─ go.sum
│     └─ main.go
├─ images/
│  └─ nginx
│     └─ Dockerfile
│  └─ backend
│     └─ werf.inc.yaml
├─ lib/
│  └─ python/
│     └─ requirements.txt
├─ openapi/
│  └─ conversions
│     ├─ testdata
│     │  ├─ v1-1.yaml
│     │  └─ v2-1.yaml
│     └─ conversions_test.go
│     └─ v2.yaml
│  └─ config-values.yaml
│  └─ doc-ru-config-values.yaml
│  └─ values.yaml
├─ templates/
│  ├─ a.yaml
│  └─ b.yaml
├─ .helmignore
├─ Chart.yaml
├─ module.yaml
├─ werf.yaml
└─ werf-giterminism.yaml

```

charts

В папке `/charts` находятся вспомогательные чарты Helm, которые используются при рендере шаблонов.

У Deckhouse Kubernetes Platform (DKP) существует собственная библиотека для работы с шаблонами – `lib-helm`. О возможностях библиотеки можно почитать в репозитории `lib-helm`. Чтобы положить библиотеку в модуль, загрузите `tgz`-архив с нужным релизом и переместите его в директорию `/charts` модуля.

crds

В этой директории лежат *CustomResourceDefinition* (CRD), которые используются компонентами модуля. CRD обновляются каждый раз, когда запускается модуль, если есть обновления.

Вложенные директории в папке `/crds` игнорируются.

Чтобы отобразить CRD из директории `/crds` в документации на сайте или модуле documentation в кластере, выполните следующие шаги:

- Создайте файл перевода со структурой аналогичной исходному файлу ресурса:
 - оставьте только параметры `description`, в которых укажите текст перевода;
 - используйте префикс `doc-ru-` в названии: например `/crds/doc-ru-crd.yaml` для `/crds/crd.yaml`.
- Создайте файлы `/docs/CR.md` и `/docs/CR.ru.md`.

docs

В папке `/docs` находится документация к модулю. Следующие вложенные директории в папке docs игнорируются при сборке документации:

- `internal`
- `internals`
- `development`
- `dev`

Следующие файлы обязательны:

- `README.md` и `README.ru.md` — описание, для чего нужен модуль, какую проблему он решает и общие архитектурные принципы.

Метаданные файла (front matter) в виде YAML-структуры должны быть во всех языковых версиях файла. Параметры, доступные для использования в метаданных:

- `title` — **(рекомендуется)** Заголовок страницы описания модуля. Пример — “Веб-консоль администратора Deckhouse”. Он же используется в навигации, если не указан параметр `linkTitle`.
- `menuTitle` — **(желательно)** Название модуля в меню слева на странице (sidebar). Пример — “Deckhouse Admin”. Если отсутствует, то используется название директории или репозитория, например `deckhouse-admin`.
- `linkTitle` — **(опционально)** Отдельный заголовок для навигации, если, например, `title` очень длинный. Если отсутствует, то используется параметр `title`.
- `description` — **(желательно)** Краткое уникальное описание содержимого страницы (до 150 символов). Не повторяет `title`. Служит продолжением названия и раскрывает его детальнее. Используется при генерации превью-ссылок и индексации поисковыми системами. Пример — «Модуль позволяет полностью управлять кластером Kubernetes через веб-интерфейс, имея только навыки работы мышью.»

▼ Пример метаданных...

```
---
title: "Веб-консоль администратора Deckhouse"
menuTitle: "Deckhouse Admin"
description: "Модуль позволяет полностью управлять кластером Kubernetes через веб-интерфейс, имея только навыки работы мышью."
---
```

Следующие файлы не обязательны, но имеют предопределённое название пункта в sidebar (меню слева) и заголовок страницы:

- `EXAMPLES.md` и `EXAMPLES.ru.md` — примеры конфигурации модуля с описанием.

Метаданные файла (front matter) в виде YAML-структуры должны быть во всех языковых версиях файла. Параметры, доступные для использования в метаданных:

- `title` — **(рекомендуется)** Заголовок страницы. Пример: “Примеры”. Он же используется в навигации, если нет `linkTitle`.
- `description` — **(желательно)** Краткое уникальное описание содержимого страницы (до 150 символов). Не повторяет `title`. Служит продолжением названия и раскрывает его детальнее. Используется при генерации превью-ссылок, индексации поисковиками. Пример: “Примеры хранения секретов в нейронной сети с автоматической подстановкой в мысли при общении.”
- `linkTitle` — **(опционально)** Отдельный заголовок для навигации, если, например, `title` очень длинный. Если отсутствует, то используется `title`.

▼ Пример метаданных...

```
---
title: "Примеры"
description: "Примеры хранения секретов в нейронной сети с автоматической подстановкой в мысли при общении."
---
```

- `FAQ.md` и `FAQ.ru.md` — часто задаваемые вопросы, касающиеся эксплуатации модуля (“Какой сценарий выбрать: А или Б?”).

Метаданные файла (front matter) в виде YAML-структуры должны быть во всех языковых версиях файла. Параметры, доступные для использования в метаданных:

- `title` – **(рекомендуется)** Заголовок страницы.
- `description` – **(желательно)** Краткое уникальное описание содержимого страницы (до 150 символов).
- `linkTitle` – **(опционально)** Отдельный заголовок для навигации, если, например, `title` очень длинный. Если отсутствует, то используется `title`.

▼ Пример метаданных...

```
---
title: "Часто задаваемые вопросы"
description: "Часто задаваемые вопросы и ответы на них."
---
```

- `ADVANCED_USAGE.md` и `ADVANCED_USAGE.ru.md` — расширенные инструкции по использованию и отладке модуля.

Метаданные файла (front matter) в виде YAML-структуры должны быть во всех языковых версиях файла. Параметры, доступные для использования в метаданных:

- `title` – **(рекомендуется)** Заголовок страницы.
- `description` – **(желательно)** Краткое уникальное описание содержимого страницы (до 150 символов).
- `linkTitle` – **(опционально)** Отдельный заголовок для навигации, если, например, `title` очень длинный. Если отсутствует, то используется `title`.

▼ Пример метаданных...

```
---
title: "Отладка модуля"
description: "В разделе разбираются все шаги по отладке модуля."
---
```

- `CR.md` и `CR.ru.md` — файлы для генерации ресурсов из папки `/crds/`. Добавьте эти файлы, если необходима генерация ресурсов из папки `/crds` модуля.

▼ Пример метаданных...

```
---
title: "Кастомные ресурсы"
---
```

- `CONFIGURATION.md` и `CONFIGURATION.ru.md` — файлы для рендеринга OpenAPI-спецификаций из файлов `/openapi/config-values.yaml` и `/openapi/doc-<LANG>-config-values.yaml`. Добавьте эти файлы, если необходима генерация таких спецификаций.

▼ Пример метаданных...

```
---
title: "Настройки модуля"
---
```

Все изображения, PDF-файлы и другие медиафайлы нужно хранить в директории `/docs` или её подкаталогах (например, `/docs/images/`). Все ссылки на файлы должны быть относительными.

Для каждого языка нужен файл с соответствующим суффиксом. Например, `image1.jpg` и `image1.ru.jpg`. Используйте ссылки:

- `[image1](image1.jpg)` в англоязычном документе;
- `[image1](image1.ru.jpg)` в русскоязычном документе.

hooks

В директории `/hooks/batch` находятся хуки модуля. Хук — это исполняемый файл, выполняемый при реакции на событие. Хуки используются модулем также для динамического взаимодействия с API Kubernetes. Например, они могут быть использованы для обработки событий, связанных с созданием или удалением объектов в кластере.

Познакомьтесь с концепцией хуков, прежде чем начать разрабатывать свой собственный хук. Для ускорения разработки хуков можно воспользоваться Go-библиотекой от команды Deckhouse.

Требования к работе хука:

- При запуске с аргументами `hook config` должна выводиться конфигурация хуков в формате

JSON.

- При запуске с аргументами `hook list` должен выводиться список всех хуков с их порядковым номером.
- При запуске с аргументами `hook run 0` должна выполняться логика хука под номером 0.

Файлы хуков должны иметь права на выполнение. Добавьте их командой `chmod +x <путь до файла с хуком>`.

В репозитории шаблона модуля можно найти примеры хуков на Go. Примеры хуков на Go также можно найти в SDK.

images

В директории `/images` находятся инструкции по сборке образов контейнеров модуля. На первом уровне находятся директории для файлов, используемых при создании образа контейнера, на втором — контекст для сборки.

Существует два способа описания образа контейнера:

1. Dockerfile — файл, который содержит команды для быстрой сборки образов. Если необходимо собрать приложение из исходного кода, поместите его рядом с Dockerfile и включите его в образ с помощью команды `COPY`.
2. Файл `werf.inc.yaml`, который является аналогом секции описания образа из `werf.yaml`.

Имя образа совпадает с именем директории для этого модуля, записанным в нотации *camelCase* с маленькой буквы. Например, директории `/images/echo-server` соответствует имя образа `echoServer`.

Собранные образы имеют content-based теги, которые можно использовать в сборке других образов. Чтобы использовать content-based теги образов, подключите библиотеку `lib-helm`. Вы также можете воспользоваться другими функциями библиотеки `helm_lib` Deckhouse Kubernetes Platform.

Пример использования content-based тега образа в Helm-чарте:

```
image: {{ include "helm_lib_module_image" (list . "<имя образа>") }}
```

openapi

conversions

В директории `/openapi/conversions` находятся файлы конверсий параметров модуля и их тесты.

Конверсии параметров модуля позволяют конвертировать OpenAPI-спецификацию параметров модуля одной версии в другую. Конверсии могут быть необходимы в случаях, когда в новой версии OpenAPI-спецификации параметр переименовывается или переносится в другое место.

Каждая конверсия возможна только между двумя смежными версиями (например с первой версии на вторую). Конверсий может быть несколько, и цепочка конверсий должна последовательно покрывать все версии спецификации параметров модуля, без “пропусков”.

Файл конверсии — это YAML-файл с именем `v<N>.yaml` или `v<N>.yml`, где `<N>` — версия конверсии. Его структура:

```
# Номер версии спецификации параметров модуля, в которую преобразовываются данные при
# конверсии.
version: N
# Набор выражений jq, которые используются при конверсии в кластере для автоматического
# преобразования
# параметров модуля предыдущей версии.
conversions: []
# Описание действий (на двух языках), которые необходимо выполнить, чтобы преобразовать
# данные
# из предыдущей версии спецификации параметров модуля.
description:
  ru: ""
  en: ""
```

Описание действий, указываемое в секции `description` файла конверсий, должно быть понятным и содержать информацию о том, какие параметры и в каком порядке нужно изменить в спецификации параметров модуля, чтобы перейти к новой версии.

Пример файла конверсии параметров модуля `v2.yaml`, где в версии 2 удаляется параметр `.auth.password`:

```
version: 2
conversions:
  - del(.auth.password) | if .auth == {} then del(.auth) end
description:
  ru: "Удалите `.auth.password`, затем `auth`, если он пуст."
  en: "Remove `.auth.password`, then `auth` if empty."
```

Тесты конверсий

Для написания тестов конверсий можно использовать функцию `conversion.TestConvert`, которой нужно передать:

- путь до исходного файла конфигурации (версия до конвертации);
- путь до ожидаемого файла конфигурации (версия после конвертации).

Пример теста конверсии.

config-values.yaml

Необходим для проверки параметров модуля, которые пользователь может настроить через ModuleConfig.

Чтобы схема была представлена в документации на сайте или в модуле documentation в кластере, создайте:

- файл `doc-ru-config-values.yaml` со структурой, аналогичной структуре файла `config-values.yaml`. В файле `doc-ru-config-values.yaml` оставьте только переведенные параметры description;
- файлы `/docs/CONFIGURATION.md` и `/docs/CONFIGURATION.ru.md` — это включит показ данных из файлов `/openapi/config-values.yaml` и `/openapi/doc-ru-config-values.yaml`.

Пример схемы `/openapi/config-values.yaml` с одним настраиваемым параметром `nodeSelector`:

```
type: object
properties:
  nodeSelector:
    type: object
    additionalProperties:
      type: string
    description: |
      The same as the Pods' `spec.nodeSelector` parameter in Kubernetes.

      If the parameter is omitted or `false`, `nodeSelector` will be determined
      automatically.</code>
```

Пример файла `/openapi/doc-ru-config-values.yaml` для русскоязычного перевода схемы:

```
properties:
  nodeSelector:
    description: |
      Описание на русском языке. Разметка Markdown.</code>
```

Валидации x-deckhouse-validations (CEL)

При разработке модуля для Deckhouse Kubernetes Platform вы можете использовать расширение OpenAPI `x-deckhouse-validations` для описания сложных правил валидации параметров модуля на языке Common Expression Language (CEL).

При использовании таких правил валидации учитывайте следующие особенности:

- Валидации можно размещать как на корневом уровне, так и внутри любого свойства (в том числе внутри объектов, массивов и `additionalProperties`).
- В выражениях доступны все параметры текущего уровня через переменную `self`.
- Прежнее значение того же уровня доступно через переменную `oldSelf`. Правила, использующие `oldSelf`, считаются правилами перехода и проверяются только при обновлении (когда на этом уровне есть прежнее значение); при первичном создании или для впервые появившегося поддерева такие правила автоматически пропускаются. Поведение совпадает с `x-kubernetes-validations`.
- Валидация работает рекурсивно: все вложенные объекты, массивы и карты также могут содержать свои `x-deckhouse-validations`.

- Поддерживаются скалярные типы, массивы, объекты и карты (`additionalProperties`).
- В случае множественных ошибок валидации пользователю будут показаны все сообщения из соответствующих правил.

Примеры правил

Ниже представлены примеры описаний сложных правил валидации на языке CEL:

- Проверка попадания значения параметра в диапазон:

```
type: object
properties:
  replicas:
    type: integer
  minReplicas:
    type: integer
  maxReplicas:
    type: integer
x-deckhouse-validations:
  - expression: "self.minReplicas <= self.replicas && self.replicas <= self.maxReplicas"
    message: "replicas должно быть между minReplicas и maxReplicas"
```

- Проверка наличия ключа:

```
- expression: "'Available' in self.stateCounts"
  message: "Должен быть ключ Available"
```

- Проверка, что хотя бы один из двух списков не пуст:

```
- expression: "(self.list1.size() == 0) != (self.list2.size() == 0)"
  message: "Ровно один из списков должен быть непустым"
```

- Проверка значения по регулярному выражению:

```
- expression: "self.details.all(key, self.details[key].matches('[a-zA-Z]*$'))"
  message: "Все значения должны содержать только буквы"
```

- Объявление поля неизменяемым (immutable) через переменную `oldSelf` (transition rule):

```
type: object
properties:
  clusterDomain:
    type: string
x-deckhouse-validations:
  - expression: "self.clusterDomain == oldSelf.clusterDomain"
    message: "Поле clusterDomain нельзя изменить после создания"
```

Правило проверяется только на обновлении: при первичном создании (когда прежнего значения ещё нет) оно пропускается автоматически — так же, как правило перехода в `x-kubernetes-validations`.

- Запрет удалять элементы из списка (список можно только дополнять):

```
- expression: "oldSelf.items.all(x, x in self.items)"
  message: "Из items нельзя удалять элементы, можно только добавлять"
```

Валидация скалярных значений и массивов

Валидации скалярных значений и массивов имеют следующие особенности:

- Если свойство — скаляр (например, число или строка), то в выражении CEL переменная `self` будет этим значением.
- Если свойство — массив, то `self` будет массивом, и можно использовать методы `.size()`, `.all()`, `.exists()` и т.д.

Пример для массива:

```
type: object
properties:
  items:
    type: array
    items:
      type: string
  x-deckhouse-validations:
    - expression: "self.size() > 0"
      message: "Список items не должен быть пустым"
```

Валидация `additionalProperties (map)`

Для объектов с `additionalProperties (map)` можно валидировать ключи и значения через методы `.all(key, ...)`, `.exists(key, ...)` и т.д.

Пример:

```
type: object
properties:
  мумар:
    type: object
    additionalProperties:
      type: integer
  x-deckhouse-validations:
    - expression: "self.all(key, self[key] > 0)"
      message: "Все значения в мумар должны быть больше 0"
```

values.yaml

Необходим для проверки исходных данных при рендере шаблонов без использования дополнительных функций Helm chart. Ближайший аналог — `schema`-файлы из Helm.

В `values.yaml` можно автоматически добавить валидацию параметров из `config-values.yaml`. В этом случае, минимальный `values.yaml` выглядит следующим образом:

```
x-extend:
  schema: config-values.yaml
type: object
properties:
  internal:
    type: object
    default: {}
```

templates

В директории `/templates` находятся шаблоны Helm.

- Для доступа к настройкам модуля в шаблонах используйте путь `.Values.<имяМодуля>`, а для глобальных настроек `.Values.global`. Имя модуля конвертируется в нотацию `camelCase`.
- Для упрощения работы с шаблонами используйте `lib-helm` – это набор дополнительных функций, которые облегчают работу с глобальными и модульными значениями.
- Доступы в registry из ресурса `ModuleSource` доступны по пути `.Values.<имяМодуля>.registry.dockercfg`.
- Чтобы использовать эти функции для пула образов в контроллерах, создайте секрет и добавьте его в соответствующий параметр: `"imagePullSecrets": [{"name": "registry-creds"}]`.

```

apiVersion: v1
kind: Secret
metadata:
  name: registry-creds
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: {{ .Values.<имяМодуля>.registry.dockercfg }}

```

Модуль может иметь параметры, с помощью которых может менять своё поведение. Параметры модуля и схема их валидации описываются в OpenAPI-схемах в директории `/openapi`.

Настройки лежат в двух файлах: `config-values.yaml` и `values.yaml`.

Пример OpenAPI-схемы можно найти в шаблоне модуля.

.helmignore

Исключите файлы из Helm-релиза с помощью `.helmignore`. В случае модулей DKP директории `/crds`, `/images`, `/hooks`, `/openapi` обязательно добавляйте в `.helmignore`, чтобы избежать превышения лимита размера Helm-релиза в 1 Мб.

Chart.yaml

Файл для чарта, аналогичный `Chart.yaml` из Helm. Должен содержать, как минимум, параметр `name` с именем модуля и параметр `version` с версией. Вы можете не создавать данный файл, Deckhouse создаст его автоматически.

Пример:

```

name: echoserver
version: 0.0.1
dependencies:
- name: deckhouse_lib_helm
  version: 1.38.0
  repository: https://deckhouse.github.io/lib-helm

```

module.yaml

Файл `module.yaml` в корне папки модуля содержит метаданные модуля.

Файл может отсутствовать, но рекомендуется его заполнить. Большинство метаданных будут доступны в объекте `Module` в кластере. Объект `Module` будет создан автоматически после настройки источника модулей (ресурс `ModuleSource`) и успешной синхронизации.

Параметры, которые можно использовать в `module.yaml`:

Параметр	Тип	Описание
<code>namespace</code>	Строка	Неймспейс, в котором будут развёрнуты компоненты модуля
<code>subsystems</code>	Массив строк	Список подсистем, к которым относится модуль
<code>accessibility</code>	Объект	Настройки доступности модуля
<code>accessibility.editions</code>	Объект	Настройки работы модуля в редакциях DKP
<code>accessibility.editions.available</code>	Булевый	Определяет доступность модуля в редакции DKP
<code>accessibility.editions.enabledInBundles</code>	Массив строк	Список наборов модулей, в которых модуль должен быть включён по умолчанию
<code>descriptions</code>	Объект	Произвольное текстовое описание назначения модуля
<code>descriptions.en</code>	Строка	Текстовое описание на английском языке

Параметр	Тип	Описание
<code>descriptions.ru</code>	Строка	Текстовое описание на русском языке
<code>disable</code>	Объект	Параметры, связанные с поведением при отключении модуля
<code>disable.confirmation</code>	Булевый	Требовать подтверждение при отключении модуля
<code>disable.message</code>	Строка	Сообщение с информацией о том, что произойдёт при отключении модуля

Если для отключения модуля требуется подтверждение (параметр `confirmation` установлен в `true`), то отключение будет возможно только в случае, если на соответствующем объекте `ModuleConfig` установлена аннотация `modules.deckhouse.io/allow-disabling=true`. Если такой аннотации нет, при попытке отключить модуль будет выведено предупреждение, включающее сообщение из параметра `message`.

Другие параметры `module.yaml`:

Параметр	Тип	Описание
<code>name</code>	Строка, обязательный параметр	Имя модуля в Kebab Case. Например, <code>echo-server</code>
<code>exclusiveGroup</code>	Строка	Если несколько модулей имеют одно и то же значение этого параметра, то только один из них может быть активен в системе одновременно. Это предотвращает конфликты между модулями, выполняющими схожие или несовместимые задачи
<code>requirements</code>	Объект	Зависимости модуля — условия, при которых Deckhouse Kubernetes Platform (DKP) может запустить модуль
<code>requirements.deckhouse</code>	Строка	Зависимость от версии Deckhouse Kubernetes Platform
<code>requirements.kubernetes</code>	Строка	Зависимость от версии Kubernetes
<code>requirements.modules</code>	Объект	Зависимость от версий других модулей
<code>stage</code>	Строка	Стадия жизненного цикла модуля. Допустимые значения: <code>Experimental</code> , <code>Preview</code> , <code>General Availability</code> , <code>Deprecated</code> . Если <code>stage</code> установлен в <code>Experimental</code> , модуль нельзя включить по умолчанию. Чтобы разрешить использовать такие модули, установите соответствующий параметр в <code>true</code>
<code>tags</code>	Массив строк	Дополнительные теги модуля. Теги преобразуются в лейблы объекта <code>Module</code> по шаблону <code>module.deckhouse.io/<TAG>=" "</code> . Например, если указать два тега <code>test</code> и <code>example</code> , то объект <code>Module</code> получит лейблы <code>module.deckhouse.io/test=" "</code> и <code>module.deckhouse.io/example=" "</code>

Параметр	Тип	Описание
<code>weight</code>	Число	Вес модуля. Влияет на порядок запуска модулей: модули с меньшим значением <code>weight</code> запускаются раньше. По умолчанию — 900. На порядок запуска также влияют зависимости модуля
<code>critical</code>	Булевый	Помечает модуль как критичный для начальной инициализации кластера. Такие модули (если они используются в кластере) запускаются в процессе начальной загрузки кластера, до того как кластер считается полностью готовым к работе. Остальные модули (не являющиеся критичными) запускаются только после того, как кластер полностью готов к работе. По умолчанию — <code>false</code>

Пример описания метаданных модуля `hello-world`:

```
name: hello-world
tags: ["test", "myTag"]
weight: 960
stage: "Experimental"
namespace: "test"
exclusiveGroup: "group"
subsystems:
  - test
  - test1
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
descriptions:
  en: "The module to say hello to the world."
  ru: "Модуль, который приветствует мир."
requirements:
  deckhouse: ">= 1.61"
  kubernetes: ">= 1.27"
disable:
  confirmation: true
  message: "Отключение этого модуля приведёт к удалению всех созданных им ресурсов."
```

Настройка доступности модуля в редакциях DKP

Параметр `accessibility` позволяет задать редакции DKP и наборы модулей (bundles), в которых будет доступен модуль, а также определить, будет ли он включаться по умолчанию.

```

accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Default
        - Managed
  ce:
    available: false
  ee:
    available: true
    enabledInBundles:
      - Managed

```

Описание параметров:

- `accessibility` — *Объект*. Корневой блок настройки доступности модуля.
- `editions` — *Объект*. Набор ключей с названиями редакций. Для каждой редакции можно задать собственные настройки доступности.
- `_default` — *Объект*. Настройка по умолчанию, если отсутствует конфигурация для конкретной редакции.
- `available` — *Булевый*. Определяет, доступен ли модуль в рамках указанной редакции.
- `enabledInBundles` — *Массив строк*. Наборы модулей, в которых модуль будет включён по умолчанию. Поддерживаемые наборы модулей (состав каждого из наборов доступен на этой странице):
 - `Default` — рекомендованный набор модулей для работы кластера. Включает средства мониторинга, контроля авторизации, организации работы сети и другие необходимые компоненты.
 - `Managed` — набор модулей для кластеров, управляемых облачными провайдерами (например, Google Kubernetes Engine).
 - `Minimal` — минимальный набор, включающий только текущий модуль.

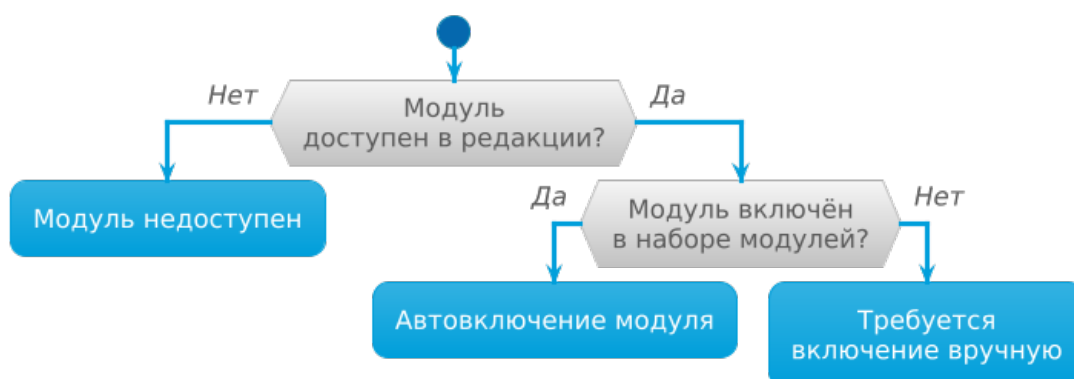


Обратите внимание, что в этот набор не входят базовые модули (например, модуль работы с CNI). Без включения базовых модулей DKP может работать только в уже развёрнутом кластере. Список модулей, которые нужно включить вручную при установке, приведён в разделе «Особенности работы с набором модулей Minimal».

- Блоки с названиями редакций. Позволяют задать поведение модуля в указанных редакциях. Возможные значения: `be`, `ce`, `ee`, `se`, `se-plus`.

Логика определения доступности модуля

Схема ниже показывает логику определения доступности и включения модуля в зависимости от настроек:



Примеры конфигурации

В следующем примере конфигурации модуль будет недоступен во всех редакциях, кроме DKP Enterprise Edition. В DKP Enterprise Edition модуль будет включён по умолчанию в наборе модулей `Managed`.


```
accessibility:
  editions:
    _default:
      available: false
  ee:
    available: true
    enabledInBundles:
      - Managed
```

В следующем примере конфигурации модуль будет доступен во всех редакциях DKP. В наборах модулей `Managed` и `Default` модуль будет включён по умолчанию.

```
accessibility:
  editions:
    _default:
      available: true
    enabledInBundles:
      - Managed
      - Default
```

В следующем примере модуль будет доступен во всех редакциях DKP. Модуль будет включён в наборах модулей `Default` и `Managed` во всех редакциях, кроме DKP Basic Edition и DKP Community Edition.

```
accessibility:
  editions:
    _default:
      available: true
    enabledInBundles:
      - Default
      - Managed
  be:
    available: false
  ce:
    available: false
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Глобальные параметры
- Параметры модуля `deckhouse`

Сборка и публикация модуля

Deckhouse Kubernetes Platform (DKP) использует хранилище образов контейнеров для загрузки и обновления модуля. В хранилище образов хранятся артефакты модуля. Артефакты модуля появляются в результате сборки модуля, после чего их можно загрузить (опубликовать) в хранилище образов.

Состав артефактов модуля

В результате сборки модуля создаются три типа артефактов, которые впоследствии загружаются в хранилище образов:

- **Образы контейнеров приложений.** Правила сборки и исходный код таких образов находятся в директории с именем *приложения* внутри `images`. Собранные образы указываются в шаблонах и запускаются в кластере. Образы тегируются `content-based` тегами, и для их использования в шаблонах необходимо подключить библиотеку `lib-helm`.
- **Образ модуля.** Правила сборки модуля находятся в файле `werf.yaml` в директории модуля. В качестве тегов образов используется семантическое версионирование.
- **Релиз.** Артефакт версии модуля. На основе данных релиза DKP принимает решение об обновлении модуля в кластере. У релизов есть два типа тегов: тег семантического версионирования (как у образа модуля) и тег, соответствующий каналу обновлений (например, `alpha`, `beta` и т. д.). В шаблоне модуля есть пример `workflow` для GitHub Actions, где релиз создается автоматически при сборке.

Сборка артефактов модуля и публикация в хранилище образов

Для сборки артефактов модуля и загрузки их в хранилище образов в рамках процесса CI/CD мы предлагаем воспользоваться подготовленными GitHub Actions.

В репозитории шаблона модуля представлен пример модуля, который содержит простой workflow для GitHub Actions и использует GitHub Packages (ghcr.io) в качестве хранилища образов. В представленном примере workflow используется следующая логика:

- Сборка артефактов модуля при изменениях в рамках PR и при слиянии изменений в ветку main.
- Сборка артефактов модуля из тегов с использованием продуктивного хранилища образов.
- Публикация модуля в хранилище образов GitHub Packages в выбранный канал обновлений из тега.

Артефакты модуля будут загружаться по адресу `ghcr.io/<OWNER>/modules/`, который будет являться источником модулей.

Выполните следующие настройки в свойствах вашего проекта на GitHub, чтобы workflow модуля работал корректно:

- Откройте страницу *Settings -> Actions -> General*.
- Установите параметр *Read and write permissions* в разделе *Workflow permissions*.

Вы также можете модифицировать workflow, предусмотреть использование своего хранилища образов и более сложный процесс сборки и публикации (например, с использованием отдельных хранилищ образов для разработки и промышленной эксплуатации).



При разработке **нескольких модулей** и их публикации в GitHub Packages необходимо использовать Personal Access Token (PAT) аккаунта.

Не используйте `GITHUB_TOKEN` в GitHub Workflows, чтобы избежать проблем с правами доступа при загрузке образов. Это связано с тем, что конечные релизные образы сохраняются по адресу `ghcr.io/<OWNER>/modules/`, принадлежащему первому созданному репозиторию.

Пример адаптации шаблона модуля для использования PAT:

- На странице *Settings -> Secrets and variables -> Actions* создайте Secret с названием `TOKEN`, содержащий PAT.
- Замените переменную `GITHUB_TOKEN` на `TOKEN` в `.github/workflows/`:

```
cd <REPO>
sed -i -e 's/GITHUB_TOKEN/TOKEN/g' $(find .github/workflows/ -type f)
```



Артефакты модуля также можно собрать локально с помощью `werf` (это может потребоваться, например, при отладке).

Вы также можете самостоятельно сделать сборку артефактов модуля и публикацию для вашей системы CI/CD по аналогии с workflow для GitHub Actions, приведенном в шаблоне модуля, но это может потребовать глубокого понимания процессов сборки и публикации модуля. Обратитесь к сообществу при появлении вопросов и затруднений.

Общий сценарий работы с workflow, приведенным в шаблоне модуля:

1. Опубликуйте изменения в коде модуля в ветке проекта на GitHub. Это запустит сборку артефактов модуля и их публикацию в хранилище образов.
2. Создайте новый релиз модуля или установите тег в формате семантического версионирования на нужном коммите.
3. Перейдите в раздел *Actions* репозитория модуля на GitHub и слева, в списке workflow, выберите *Deploy*.
4. В правой части страницы нажмите на выпадающий список *Run workflow*, выберите необходимый канал обновлений и укажите нужный тег в поле ввода тега. Нажмите кнопку *Run workflow*.
5. После успешного выполнения workflow в хранилище образов появится новая версия модуля. Опубликованную версию модуля можно использовать в кластере.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Запуск и проверка модуля в кластере

В этом разделе рассмотрен процесс запуска модуля в кластере Deckhouse Kubernetes Platform (DKP), а также подключение Deckhouse Module Tools для проверки модуля и сбора метрик.

Запуск модуля в кластере DKP

Чтобы запустить модуль в кластере, необходимо выполнить следующие шаги:

- Определить источник модулей (ресурс `ModuleSource`).
- *(не обязательно)* Определить политику обновления модуля (ресурс `ModuleUpdatePolicy`).
- Включить модуль в кластере (ресурс `ModuleConfig`).

Источник модулей

Deckhouse Kubernetes Platform (DKP) может работать со следующими видами модулей:

- Встроенные модули. Входят в состав DKP. Релизный цикл привязан к релизному циклу DKP.
- Модули из источника модулей. Релизный цикл таких модулей не привязан к релизному циклу DKP.

Чтобы указать, откуда кластеру получать информацию о модулях, создайте ресурс `ModuleSource`. В нем задаются адрес хранилища образов контейнеров, из которого DKP будет загружать модули, параметры аутентификации и другие настройки доступа.

Пример ресурса `ModuleSource`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.com/deckhouse/modules
    dockerCfg: <base64 encoded credentials>
```

После создания ресурса `ModuleSource` DKP начнет выполнять периодическую (раз в три минуты) синхронизацию данных с источником модулей (загружать информацию о модулях, доступны в источнике).

Проверить состояние синхронизации можно с помощью следующей команды:

```
d8 k get ms
```

Пример вывода в случае успешной синхронизации:

```
$ d8 k get ms
NAME      COUNT  SYNC  MSG
example    2     16s  Ready
```

В случае ошибок синхронизации в столбце `MSG` будет указано общее описание ошибки. Пример:

```
$ d8 k get ms
NAME      COUNT  SYNC  MSG
example    2     16s  Some errors occurred. Inspect status for details
```

Подробную информацию об ошибках можно получить в поле `pullError` в статусе ресурса `ModuleSource`.

Пример получения подробной информации об ошибках из источника модулей `example`:

```
$ d8 k get ms example -o jsonpath='{range .status.modules[*]}{.name}{" module error:\n\t"}{.pullError}{"\n"}{end}}'
module-1 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-1/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
module-2 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-2/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
```

В случае успешной синхронизации, поле `.status.modules` ресурса `ModuleSource` будет содержать список модулей, доступных для включения в кластере.

Пример получения списка модулей, доступных из источника модулей `example`:

```
$ d8 k get ms example -o jsonpath='{.status.modules[*].name}'
module-1 module-2
```

Полный список модулей, доступных из всех созданных в кластере источников модулей, можно получить с помощью следующей команды:

```
d8 k get ms -o jsonpath='{.items[*].status.modules[*].name}'
```

После создания ресурса ModuleSource и успешной синхронизации, в кластере должны начать появляться *модули* — ресурсы Module (DKP создает их автоматически, создавать их не нужно). Посмотреть список модулей можно с помощью следующей команды:

```
d8 k get module
```

Пример получения списка модулей:

```
$ d8 k get module
```

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
module-one			Available	False	False
module-two			Available	False	False

Чтобы получить дополнительную информацию о модуле, выполните следующую команду:

```
d8 k get module module-one -oyaml
```

Пример вывода:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-12-12T10:49:40Z"
  generation: 1
  name: module-one
  resourceVersion: "241504954"
  uid: 3ae75474-8e96-4105-a939-6df71cba82d8
properties:
  availableSources:
    - example
status:
  conditions:
    - lastProbeTime: "2024-12-12T10:49:41Z"
      lastTransitionTime: "2024-12-12T10:49:41Z"
      message: disabled
      reason: Disabled
      status: "False"
      type: EnabledByModuleConfig
    - lastProbeTime: "2024-12-12T10:49:41Z"
      lastTransitionTime: "2024-12-12T10:49:41Z"
      status: "False"
      type: EnabledByModuleManager
    - lastProbeTime: "2024-12-16T15:46:26Z"
      lastTransitionTime: "2024-12-12T10:49:41Z"
      message: not installed
      reason: NotInstalled
      status: "False"
      type: IsReady
  phase: Available
```

В Module указаны доступные источники из которых его можно скачать (в примере он только один).

Далее нужно включить модуль. Для этого нужно создать ModuleConfig с названием модуля.

За включение модуля отвечает параметр `enabled` ModuleConfig. Если модуль доступен из нескольких источников (ресурс ModuleSource), необходимый источник можно указать в параметре `source`.

Политику обновления (имя `ModuleUpdatePolicy`) можно указать в параметре `updatePolicy`. Политику обновления можно не указывать, — в этом случае она будет унаследована от параметров обновления Deckhouse Kubernetes Platform (`releaseChannel` и `update ModuleConfig deckhouse`).

Пример `ModuleConfig` для включения модуля `module-one` из источника `example`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-one
spec:
  enabled: true
  source: example
```

⚠ Если в конфигурации модуля есть обязательные параметры, и модуль включён без их указания, произойдёт ошибка валидации конфигурации. В этом случае сработает алерт `D8DeckhouseModuleValidationError`, а модуль не будет успешно активирован.

Для получения подробной информации используйте команду:

```
d8 k get mr -l module=<MODULE_NAME>
```

Убедитесь, что вы указываете необходимые параметры конфигурации в `ModuleConfig` согласно документации модуля.

После включения модуля, он должен перейти в фазу скачивания (`Downloading`):

```
$ d8 k get module module-one
NAME      STAGE    SOURCE    PHASE      ENABLED  READY
module-one  example Downloading False      False
```

⚠ Если модуль не перешел в фазу скачивания, проверьте источник модуля (`ModuleSource`), возможно модуль не может скачаться.

После успешного скачивания модуль перейдет в фазу установки (`Installing`):

```
$ d8 k get module module-one
NAME      STAGE    SOURCE    PHASE      ENABLED  READY
module-one  example Installing False      False
```

Если модуль успешно установился, то он перейдет в фазу готовности (`Ready`):

```
$ d8 k get module module-one
NAME      STAGE    SOURCE    PHASE    ENABLED  READY
module-one  example Ready     True     True
```

Пример объекта `Module` в кластере, когда модуль успешно установился:

```

apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 1
  name: module-one
  resourceVersion: "242153004"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
    - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernetes: '> v1.25.0'
  source: example
  version: v0.7.24
  weight: 910
status:
  conditions:
    - lastProbeTime: "2024-12-12T15:49:35Z"
      lastTransitionTime: "2024-12-12T15:49:35Z"
      status: "True"
      type: EnabledByModuleConfig
    - lastProbeTime: "2024-12-17T09:35:27Z"
      lastTransitionTime: "2024-12-12T15:49:39Z"
      status: "True"
      type: EnabledByModuleManager
    - lastProbeTime: "2024-12-17T09:35:27Z"
      lastTransitionTime: "2024-12-17T09:35:25Z"
      status: "True"
      type: IsReady
    - lastProbeTime: "2024-12-17T09:32:50Z"
      lastTransitionTime: "2024-12-17T09:32:50Z"
      status: "False"
      type: IsOverridden
  hooksState: 'v0.7.24/hooks/moduleVersion.py: ok'
  phase: Ready

```

В Module можно увидеть текущую установленную версию модуля, его вес, источник откуда он скачался, зависимости и релизный канал.

При возникновении каких либо ошибок, модуль перейдет в фазу ошибки (**Error**):

```

$ d8 k get module module-one
NAME          STAGE    SOURCE    PHASE    ENABLED    READY
module-one    example  Error     True     Error

```

Если у включенного модуля есть несколько доступных источников, и в его ModuleConfig явно не выбран источник модуля, модуль перейдет в фазу конфликта (**Conflict**):

```

$ d8 k get module module-one
NAME          STAGE    SOURCE    PHASE    ENABLED    READY
module-one    Conflict False     False

```

Чтобы разрешить конфликт, укажите источник модуля (имя ModuleSource) явно в ModuleConfig.

После скачивания модуля в кластере появятся релизы модуля — объекты ModuleRelease.

Посмотреть список релизов можно с помощью следующей команды:

```
d8 k get mr
```

Пример получения списка релизов модулей:

```
$ d8 k get mr
```

NAME	PHASE	UPDATE POLICY	TRANSITIONTIME	MESSAGE
module-one-v0.7.23	Superseded	deckhouse	33h	
module-one-v0.7.24	Deployed	deckhouse	33h	
module-two-v1.2.0	Superseded	deckhouse	48d	
module-two-v1.2.1	Superseded	deckhouse	48d	
module-two-v1.2.3	Deployed	deckhouse	48d	
module-two-v1.2.4	Superseded	deckhouse	44d	
module-two-v1.2.5	Pending	deckhouse	44d	Waiting for the 'release.deckhouse.io/approved: \"true\"' annotation

Если релиз модуля находится в статусе `Superseded`, это значит что релиз модуля устарел, и есть более новый релиз, который его заменил.



Если релиз модуля находится в статусе `Pending`, то это значит что он требует ручного подтверждения для установки (смотри далее про политику обновления модуля). Подтвердить релиз модуля можно следующей командой (укажите имя `moduleRelease`):

```
d8 k annotate mr <module_release_name> modules.deckhouse.io/approved="true"
```

Для удобства это можно сделать с помощью `d8 CLI` (имена модулей и версии автодополняются):

```
d8 system module approve <module-name> <version>
```

Переключение модуля на другой источник модулей

Если необходимо развернуть модуль из другого источника модулей, выполните следующие шаги:

1. Создайте новый ресурс `ModuleSource`.
2. Укажите его в поле `source` в `ModuleConfig`.
3. Проверьте, что новые релизы модуля (объекты `ModuleRelease`) создаются из нового источника модулей в соответствии с политикой обновления:

```
d8 k get mr
```

Политика обновления модуля

Политика обновления модуля — это правила, по которым DKP обновляет модули в кластере. Она определяется ресурсом `ModuleUpdatePolicy`, в котором можно настроить:

- режим обновления модуля (автоматический, ручной, обновления отключены);
- канал стабильности, используемый при обновлении;
- окна автоматического обновления, в пределах которых разрешено обновление модуля.

Создавать ресурс `ModuleUpdatePolicy` не обязательно. Если политика обновления для модуля не определена (отсутствует соответствующий ресурс `ModuleUpdatePolicy`), то настройки обновления соответствуют настройкам обновления самого DKP (параметр `update` модуля `deckhouse`).

Пример ресурса `ModuleUpdatePolicy`, политика обновления которого разрешает автоматическое обновление модуля по понедельникам и средам с 13:30 до 14:00 UTC:

```

apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
      - days:
        - "Mon"
        - "Wed"
      from: "13:30"
      to: "14:00"

```

Политика обновления указывается в поле `updatePolicy` в `ModuleConfig`.

Включение модуля в кластере

Прежде чем включить модуль, проверьте что он доступен для включения. Выполните следующую команду, чтобы вывести список всех доступных модулей DKP:

```
d8 k get modules
```

Модуль должен быть в списке.

Пример вывода:

```

$ d8 k get module
NAME          STAGE    SOURCE    PHASE      ENABLED  READY
...
module-one
module-two
...

```

NAME	STAGE	SOURCE	PHASE	ENABLED	READY
...					
module-one			Available	False	False
module-two			Available	False	False
...					

Вывод показывает, что модуль `module-one` доступен для включения.

Если модуля нет в списке, то проверьте что определен источник модулей и модуль есть в списке в источнике модулей. Также проверьте политику обновления модуля (если она определена). Если политика обновления модуля не определена, то она соответствует политике обновления DKP (параметр `releaseChannel` и секция `update` параметров модуля `deckhouse`).

Включить модуль можно аналогично встроенному модулю DKP любым из следующих способов:

- Выполнить следующую команду (укажите имя модуля):

```
d8 system module enable <MODULE_NAME>
```

- Создать ресурс `ModuleConfig` с параметром `enabled: true` и настройками модуля.

Пример `ModuleConfig`, для включения и настройки модуля `module-one` в кластере:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-one
spec:
  enabled: true
  settings:
    parameter: value
  version: 1

```

Если что-то пошло не так

Если при включении модуля в кластере возникли ошибки, то получить информацию о них можно следующими способами:

- Посмотреть журнал Deckhouse:

```
d8 k -n d8-system logs -l app=deckhouse
```

- Посмотреть объект Module подробнее:

```
d8 k get module module-one -oyaml
```

- Посмотреть объект ModuleConfig модуля:

Пример вывода информации об ошибке модуля `module-one`:

```
$ d8 k get moduleconfig module-one
NAME          ENABLED  VERSION  AGE    MESSAGE
module-one    true          7s      Ignored: unknown module name
```

- Посмотреть объект ModuleSource:

Пример вывода если у источника модуля есть проблемы со скачиванием модуля:

```
$ d8 k get ms
NAME          COUNT  SYNC  MSG
example       2      16s   Some errors occurred. Inspect status for details
```

По аналогии с `DeckhouseRelease` (ресурсом релиза DKP) у модулей есть аналогичный ресурс — `ModuleRelease`. DKP создает `ModuleRelease` исходя из того, что хранится в хранилище образов. При поиске проблем с модулем проверьте также доступные в кластере `ModuleRelease`:

```
d8 k get mr
```

Пример вывода:

```
$ d8 k get mr
NAME                PHASE          UPDATE POLICY          TRANSITIONTIME  MESSAGE
module-1-v1.23.2    Pending        example-update-policy   3m              Waiting for
the 'release.deckhouse.io/approved: "true"' annotation
```

В примере вывода показан `ModuleRelease`, когда режим обновления (параметр `update.mode` `ModuleUpdatePolicy` установлен в `Manual`). В этом случае необходимо вручную подтвердить установку новой версии модуля, установив на `ModuleRelease` аннотацию `modules.deckhouse.io/approved="true"`:

```
d8 k annotate mr module-1-v1.23.2 modules.deckhouse.io/approved="true"
```

Для удобства это можно сделать с помощью `d8` CLI (имена модулей и версии автодополняются):

```
d8 system module approve module-1 v1.23.2
```

Подключение Deckhouse Module Tools для проверки модуля

Для автоматической проверки структуры модуля и, при необходимости, отправки статистики, в сборку можно подключить `Deckhouse Module Tools` (DMT).

Для GitHub-проектов

Для GitHub доступен отдельный GitHub Action для подключения DMT к модулю.

Чтобы подключить DMT, в конфигурации workflow сборки `[project].github/workflows/build.yml` добавьте шаг для выполнения проверки:

```
jobs:
  lint:
    runs-on: ubuntu-latest
    continue-on-error: true
    name: Linting
    steps:
      - uses: actions/checkout@v4
      - uses: deckhouse/modules-actions/lint@main
    env:
      DMT_METRICS_URL: ${ secrets.DMT_METRICS_URL }
      DMT_METRICS_TOKEN: ${ secrets.DMT_METRICS_TOKEN }
```

Переменные `DMT_METRICS_URL` и `DMT_METRICS_TOKEN` - необязательные. При их наличии DMT будет отправлять телеметрию на указанный адрес.

i Если модуль находится в GitHub-группе `deckhouse`, значения этих переменных будут автоматически получены из настроенных секретов.

Пример полной конфигурации можно увидеть в файле `build_dev.yml`.

Для упрощения настройки сборки воспользуйтесь шаблонами конфигурации.

Для GitLab-проектов

Для GitLab также доступны готовые шаблоны, которые можно подключить в `.gitlab-ci.yml` для автоматической настройки процессов сборки и проверки корректности:

- **Setup:** Шаблон конфигурации для настройки.
- **Build:** Шаблон конфигурации для процесса сборки.

Шаги для подключения

1. В файле `.gitlab-ci.yml` вашего проекта добавьте ссылки на шаблоны:

```
include:
  - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-ci/refs/heads/main/templates/Setup.gitlab-ci.yml
  - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-ci/refs/heads/main/templates/Build.gitlab-ci.yml
```

2. После подключения шаблонов, в той же конфигурации `.gitlab-ci.yml` добавьте шаг для выполнения проверки:

```
Lint:
  extends: .lint
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `deckhouse`

Зависимости модуля Deckhouse Kubernetes Platform

В этом разделе описаны зависимости, которые могут быть установлены для модуля.

Зависимости — это набор условий (требований), которые должны выполняться, чтобы Deckhouse Kubernetes Platform (DKP) мог запустить модуль.

DKP поддерживает следующие зависимости для модуля:

- зависимость от версии Deckhouse Kubernetes Platform;
- зависимость от версии Kubernetes;
- зависимость от версии других модулей.

Зависимость от версии Deckhouse Kubernetes Platform

Эта зависимость определяет минимальную или максимальную версию DKP, с которой совместим

модуль.

Пример настройки зависимости модуля от версии DKP 1.61 и выше в файле `module.yaml`:

```
name: test
weight: 901
requirements:
  deckhouse: ">= 1.61"
```

i Для тестирования можно задать переменную окружения `TEST_EXTENDER_DECKHOUSE_VERSION`, чтобы симулировать желаемую версию DKP.

Зависимость проверяется в следующих случаях:

1. **При установке или обновлении модуля.** Если версия DKP не соответствует требованиям, указанным в зависимостях модуля релиза, его установка или обновление не будут выполнены.

Пример ресурса `ModuleRelease`, когда версия DKP не соответствует требованиям модуля:

```
root@dev-master-0:~# d8 k get mr
```

Выводимая информация:

NAME	PHASE	UPDATE POLICY	TRANSITIONTIME	MESSAGE
test-v0.8.3	Pending	test-alpha	2m30s	requirements are not satisfied: current deckhouse version is not suitable: 1.0.0 is less than or equal to v1.64.0

2. **При обновлении DKP.** Проверяется, соответствует ли новая версия DKP зависимостям установленных и активных модулей. Если хотя бы один модуль несовместим с новой версией, обновление DKP не выполнится.

Пример ресурса `DeckhouseRelease`, когда версия DKP не соответствует требованиям модуля:

```
root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
```

Выводимая информация:

NAME	PHASE	TRANSITIONTIME	MESSAGE
v1.73.3	Skipped	74m	
v1.73.4	Pending	2m13s	requirements of test are not satisfied: v1.73.4 deckhouse version is not suitable: v1.73.4 is greater than or equal to v1.73.4

3. **При первичном анализе модулей.** Проверяются текущая версия DKP и зависимости уже установленных модулей. Если обнаружено несоответствие, модуль будет отключён.

Зависимость от версии Kubernetes

Эта зависимость определяет минимальную или максимальную версию Kubernetes, с которой совместим модуль.

Пример настройки зависимости от Kubernetes 1.28 и выше в файле `module.yaml`:

```
name: test
weight: 901
requirements:
  kubernetes: ">= 1.28"
```

i Для тестирования можно задать переменную окружения `TEST_EXTENDER_KUBERNETES_VERSION`, чтобы симулировать желаемую версию Kubernetes.

Зависимость проверяется в следующих случаях:

1. **При установке или обновлении модуля.** Если версия Kubernetes не соответствует требованиям, указанным в зависимостях модуля релиза, установка или обновление не будут выполнены.

Пример ресурса `ModuleRelease`, когда версия Kubernetes не соответствует требованиям модуля:

```
root@dev-master-0:~# d8 k get modulereleases.deckhouse.io
```

Выводимая информация:

NAME	PHASE	UPDATE POLICY	TRANSITIONTIME	MESSAGE
test-v0.8.2	Pending	test-alpha	24m	requirements are not
satisfied: current kubernetes	version is not suitable: 1.29.6 is less than or equal to 1.29			
virtualization-v.0.0.0-dev4	Deployed	deckhouse	142d	

2. **При обновлении версии Kubernetes.** Проверяются зависимости активных модулей, и если хотя бы один модуль несовместим с новой версией Kubernetes, изменение версии не будет принято.

Пример вывода при несовместимости модуля с новой версией Kubernetes:

```
root@dev-master-0:~# d8 system edit cluster-configuration
```

Выводимая информация:

```
Save cluster-configuration back to the Kubernetes cluster
Update cluster-configuration secret
Attempt 1 of 5 |
    Update cluster-configuration secret failed, next attempt will be in 5s"
    Error: admission webhook "kubernetes-version.deckhouse-webhook.deckhouse.io" denied the request:
    requirements of test are not satisfied: 1.27 kubernetes version is not suitable: 1.27.0 is less than or
    equal to 1.28
```

3. **При первичном анализе модулей.** Если версия Kubernetes не соответствует зависимостям уже установленных модулей, DKP отключит такие модули.
4. **При обновлении DKP.** Проверяется значение версии Kubernetes, установленной по умолчанию для DKP, если оно несовместимо с активными модулями, обновление DKP не будет выполнено.

Пример ресурса DeckhouseRelease, когда версия Kubernetes не соответствует требованиям модуля:

```
root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
```

Выводимая информация:

NAME	PHASE	TRANSITIONTIME	MESSAGE
v1.73.3	Pending	7s	requirements of test are not satisfied: 1.27
kubernetes version is not suitable: 1.27.0 is less than or equal to 1.28			

Зависимость от версии других модулей

Зависимости от версии других модулей описывают условия включения, обновления и выключения модуля. Модуль в Deckhouse Kubernetes Platform может иметь обязательные и условные зависимости от версий других модулей.

Обязательные зависимости

Обязательная зависимость определяет список **включенных** модулей и их версии, которые необходимы для работы модуля.



Версия встроенного модуля DKP считается равной версии DKP.

Если необходимо указать, чтобы какой-то модуль был просто включен, не важно какой версии, то можно использовать следующий синтаксис (на примере модуля `user-authn`):

```
requirements:
  modules:
    user-authn: ">= 0.0.0"
```

Пример настройки зависимости от трех модулей:

```
name: hello-world
requirements:
  modules:
    ingress-nginx: '> 1.67.0'
    node-local-dns: '>= 0.0.0'
    operator-trivy: '> v1.64.0'
```

Условные зависимости



Условные зависимости для модулей доступны для Deckhouse Kubernetes Platform, начиная с версии 1.73. Если вы планируете использовать их для какого-то модуля, задайте для него зависимость от версии DKP 1.73 и выше.

Условная зависимость используется, когда модуль работает самостоятельно, но может использоваться совместно с другим модулем, **если он включён**.



Условные зависимости могут влиять на возможность включения, выключения и обновления обоих модулей: зависимого и того, от которого он зависит.

Чтобы указать, что зависимость является условной, добавьте `!optional` к строке ограничения версии модуля, от которого может зависеть целевой модуль:

```
name: prometheus
requirements:
  modules:
    test: ">v0.22.1 !optional"
```



Ниже рассмотрены ограничения при использовании условных зависимостей модуля и примеры настроек, где:

- `prometheus` — целевой модуль, для которого задается условная зависимость;
- `test` — модуль, совместно с которым может использоваться целевой.

Ограничения по включению и выключению prometheus при наличии условной зависимости от test

При наличии условной зависимости от версии другого модуля целевой имеет следующие ограничения по включению и выключению:

1. Если в кластере не включен `test`, `prometheus` может быть включен.

Пример: `test` выключен + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → `prometheus` будет включен, требование пропускается.

2. Если в кластере уже включен `test`, включение `prometheus` с `requirements` возможно только, если в `requirements` указаны требования, которым соответствует текущая версия `test`.

Пример: в кластере включен `test` версии `v0.21.1` + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → установка/включение `prometheus` завершится ошибкой о несоответствии зависимости (текущая версия `test` не соответствует `requirements`).

3. Если `test` будет выключен в кластере, `prometheus` останется включенным.

Пример: в кластере включен `test` + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → выключение `test` пройдет успешно, `prometheus` выключен не будет.

Ограничения по обновлению prometheus при наличии условной зависимости от test

При наличии условной зависимости от версии другого модуля целевой модуль имеет следующие ограничения по обновлению:

1. `prometheus` может быть обновлен, даже если в кластере нет модуля `test`.

Пример: `test` выключен + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → `prometheus` будет обновлен.

- Обновление для `prometheus` будет заблокировано, пока в кластере не обновится `test` указанной в `requirements` версии.

Пример: `test` включен + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → `prometheus` не будет обновлен, пока не обновится `test`.

Ограничения по включению `test`, версия которого указана в зависимостях `prometheus`

Если модуль `prometheus` включен, невозможно включить `test`, версия которого не соответствует выражению, указанному в `requirements` для `prometheus`.

Пример: `prometheus` включен + для `prometheus` задана условная зависимость `test: ">v0.22.1 !optional"` → попытка включения `test v0.21.1` завершится ошибкой о несоответствии зависимости (версия `test v0.21.1` не соответствует условию в `requirements` для `prometheus`).

Ограничения по обновлению `test`, версия которого указана в зависимостях `prometheus`

Если `prometheus` и `test` включены в кластере, обновление `test` возможно только на версию которая соответствует требованиям, указанным в `requirements` для `prometheus`.

Пример: Модули `prometheus` и `test` включены в кластере + для `prometheus` задана условная зависимость `test: "=v0.22.1 !optional"` + попытка обновления `test` до версии `0.23.1` → `test` не будет обновлен, т.к. требуемая версия не соответствует `requirements` для `prometheus`.



- Включение и отключение модулей может занимать больше времени из-за дополнительных проверок экстендера.
- Известное ограничение: во время обработки модулей список включённых модулей может кратковременно быть пустым. В редких случаях это позволяет ошибочно пройти проверку условной зависимости. Если столкнулись, повторите операцию.

Версионирование модуля

В Deckhouse Kubernetes Platform (DKP) используется принцип семантического версионирования для модулей.

При выборе номера версии используйте следующие рекомендации:

- изменение **патч-версии** (например, с `0.0.1` на `0.0.2`) — исправление дефекта;
- изменение **минорной версии** (например, с `0.0.1` на `0.1.0`) — добавление новой функции;
- изменение **мажорной версии** (например, с `0.0.1` на `1.0.0`) — добавление функции, которая кардинально меняет возможности модуля; масштабное изменение интерфейса или завершение крупного этапа работы.

Перед номером версии в теге `git` и контейнере `registry` **всегда** добавляется буква `"v"`. Например: `v0.0.73`, `v1.0.0`.

Каналы обновлений

Каналы обновлений позволяют выпускать новую версию модуля постепенно, сначала для ограниченного числа пользователей, а затем для более широкой аудитории. Вы самостоятельно определяете, насколько стабильна новая версия и на какой канал обновлений её можно опубликовать.

Важно понимать, что выбор канала обновлений не определяет стабильность модуля в целом. Каналы являются инструментом доставки обновлений и определяют степень стабильности конкретного релиза.

При публикации новой версии модуля на канал обновлений сначала используйте канал обновлений `Alpha`. Далее, если работа версии модуля не вызывает нареканий, публикуйте версию модуля последовательно на другие каналы обновлений, с учетом их стабильности: `Alpha` → `Beta` → `Early Access` → `Stable` → `Rock Solid`. Если версия модуля требует исправления ошибок, то публикация такой версии должна быть остановлена. После выпуска версии с исправлениями, необходимо повторить этап публикации версии начиная с канала обновлений `Alpha`.

Порядок публикации новой версии

Рекомендуемая последовательность публикации версии модуля в каналах обновлений:

- Опубликуйте новую версию модуля в канале `Alpha`.
- Если версия работает стабильно, последовательно публикуйте ее в следующих каналах: `Beta` → `EarlyAccess` → `Stable` → `RockSolid`.
- При возникновении ошибок остановите публикацию и исправьте их.
- Повторите публикацию версии, начиная с канала `Alpha`.

Жизненный цикл модуля

За время своего жизненного цикла модуль проходит следующие стадии:

- **Experimental** — экспериментальная версия. Функциональность модуля может сильно измениться. Совместимость с будущими версиями не гарантируется.

i Модули на стадии `Experimental` по умолчанию включить нельзя. Чтобы разрешить использовать такие модули установите параметр `allowExperimentalModules` в `true`.

- **Preview** — предварительная версия. Функциональность модуля может измениться, но основные возможности сохранятся. Совместимость с будущими версиями обеспечивается, но может потребоваться миграция.
- **General Availability (GA)** — общедоступная версия. Модуль готов к использованию в `production`-средах.
- **Deprecated** — модуль устарел. Дальнейшее развитие и поддержка модуля прекращены.

Определение стабильности модуля

В зависимости от стадии жизненного цикла модуля и канала обновлений, из которого была установлена версия модуля, общую стабильность можно определить по следующей таблице:

Стадия жизненного цикла	Каналы обновлений				
	Alpha	Beta	Early Access	Stable	Rock Solid
Experimental	Эксперименты	Эксперименты	Эксперименты	Опытная эксплуатация	Опытная эксплуатация
Preview	Эксперименты	Ограниченная эксплуатация	Ограниченная эксплуатация	Промышленная эксплуатация	Промышленная эксплуатация
General Availability	Эксперименты	Ограниченная эксплуатация	Ограниченная эксплуатация	Промышленная эксплуатация	Промышленная эксплуатация в ответственных системах
Deprecated	Отказ от использования	Отказ от использования	Отказ от использования	Отказ от использования	Отказ от использования

- **Эксперименты** — проверка функциональности, эксперименты и тестирование.
- **Опытная эксплуатация** — проверка функциональности, эксперименты и тестирование. Точечное использование опытными пользователями в окружениях, приравненных к `production`.
- **Ограниченная эксплуатация** — окружения разработки, пилотные проекты, малозначимые `production`-окружения.
- **Промышленная эксплуатация** — `production`-окружения и приравненные к ним.
- **Промышленная эксплуатация в ответственных системах** — критически важные `production`-окружения и приравненные к ним.
- **Отказ от использования** — необходимо выводить из использования.

Выводы:

- Модуль на стадии `Experimental` на канале `Stable` рекомендовано использовать в `production`-средах только ограниченно.
- Модуль на стадии `General Availability` на канале `Alpha` также не рекомендуется использовать в `production`-средах.
- Для `production`-сред подходят только модули, находящиеся на стадии `General Availability`, установленные из каналов `Early Access`, `Stable` или `Rock Solid`.
- Модули на стадии `Deprecated` рекомендуется заменить.

Версионирование API

Модули в DKP используют кастомные ресурсы для взаимодействия с пользователями. Параметр `apiVersion` с версией API этих ресурсов обновляется в соответствии со следующими правилами:

- `v1alphaX` — недавно опубликованный API. Требуется проверка удобства использования, а также структуры и корректности настроек.
- `v1betaX` — API прошел первичное тестирование. Продолжается его логическое развитие и доработка.

- `v1stableX` — стабильная версия API. С этого момента его поля не удаляются из спецификации и правила валидации не меняются в сторону большей строгости.

При необходимости можно выпустить новую версию API `v2`, которая проходит те же этапы, но с префиксом `v2`. Важно помнить, что после выпуска версии `v1stableX` Kubernetes будет считать её более приоритетной, чем `alpha`- или `beta`-версии, до выпуска новой стабильной версии `v2stableX`. При выполнении команд `kubectl apply` и `kubectl edit` по умолчанию будет использоваться версия `v1stableX`.

Выпуск новой версии API

Причины для выпуска новой версии:

- изменение структуры;
- обновление устаревших параметров.

Добавлять новые параметры можно без изменения версии.

Автоматическая конвертация API

Для автоматической конвертации параметров модуля из одной версии в другую включите в модуль соответствующие конверсии. Это может понадобиться, например, при переименовании или перемещении параметра в новой версии OpenAPI-спецификации.

Рекомендации по выпуску новых версий CRD

При выходе новой версии CustomResourceDefinition (CRD) используйте следующие рекомендации:

- Установите предыдущим версиям CRD параметр `deprecated: true`. За подробностями о работе с устаревшими версиями CRD обратитесь к документации Kubernetes.
- Не меняйте `storage`-версию, в которой данные хранятся внутри etcd, пока не пройдет как минимум два месяца после выхода новой версии.

Дополнительные ресурсы

- Параметры модуля `deckhouse`

Пример адаптации существующего чарта

Пример создания модуля `helloworld` на основе шаблона модуля и адаптации Helm-чарта `hello-world`.

Подготовка исходного кода модуля и сборки

1. Установите необходимые утилиты.
 - `git`
 - `sed`
 - `yq`
2. Сделайте форк или скопируйте репозиторий шаблона модуля.

```
git clone git@github.com:deckhouse/modules-template.git helloworld \
&& cd helloworld
```

3. Укажите имя модуля в файле `module.yaml`.

В примере будет использоваться имя модуля `helloworld`, но вы можете выбрать свое, заменив `helloworld` в командах и в названии вашего репозитория.

i Обратите внимание, что в некоторых местах в примере имя модуля может быть записано в разных форматах — *kebab-case* или *camelCase*. Если вы используете свое имя модуля, то учитывайте изменение формата имени модуля в приводимых командах.

Выполните следующую команду, чтобы указать имя модуля в файле `module.yaml`, либо отредактируйте его вручную:

```
sed -i -e 's/^name:.*$/name: helloworld/' module.yaml
```

4. Склонируйте исходный код чарта `hello-world` во временную директорию.

```
git clone https://github.com/giantswarm/hello-world-app .tmp-chart
```


5. Скопируйте шаблоны чарта в директорию `templates` модуля, предварительно очистив ее.

```
rm -rf templates/*
cp -fR .tmp-chart/helm/hello-world/templates/* templates/
cp .tmp-chart/helm/hello-world/values.yaml values.yaml
```

6. Замените в шаблонах чарта путь `.Values` на `.Values.helloworld`.

i Это архитектурная особенность `addon-operator`, ей необходимо следовать для обращения к `values` модуля.

```
sed -i -e 's/.Values/.Values.helloworld/g' $(find templates/ -type f)
```

7. Добавьте OpenAPI-схему настроек модуля.

Параметры модуля указываются в OpenAPI-схеме в директории `openapi`. Выполните следующую команду, чтобы преобразовать JSON-схему параметров чарта в OpenAPI-схему модуля:

```
jq 'walk(
  if type == "object" and .type == "object" and (keys | length) == 1
  then . + {additionalProperties: true}
  else .
  end
)' .tmp-chart/helm/hello-world/values.schema.json > openapi/config-values.yaml
```

8. Опишите правило сборки образа контейнера приложения.

Правила сборки образов контейнеров приложений должны находиться в подкаталоге директории `images` модуля. Выполните следующую команду, чтобы создать директорию образа приложения и `Dockerfile` с правилами сборки образа.

```
rm -rf images/*
mkdir images/helloworld
echo "FROM quay.io/giantswarm/helloworld:0.2.0" > images/helloworld/Dockerfile
```

9. Замените образ в манифесте Deployment на хелпер библиотеки Deckhouse Kubernetes Platform. Это позволит использовать актуальный `content-based`-тэг образа.

```
sed -Ei 's/image\:(.*)/image: {{ include "helm_lib_module_image" (list .
"helloworld") }}/g' templates/deployment.yaml
```

10. Удалите хуки модуля, CRD и временные файлы.

Пример не использует хуки и `CustomResourceDefinition`. Выполните следующие команды, чтобы очистить папки `hooks` и `crds`:

```
rm -rf hooks/
rm -rf crds/
rm -rf .tmp-chart
```

11. Настройте CI/CD.

В шаблоне проекта в директории `.github` находятся готовые файлы workflow GitHub Actions, которые реализуют простую схему сборки и публикации модуля с использованием хранилища образов GitHub Packages (`ghcr.io`). Артефакты модуля будут загружаться по адресу `ghcr.io/<OWNER>/modules/`, который будет являться источником модулей. Внесите изменения в файлы workflow, если вам не подходит предложенный вариант.

Выполните следующие настройки в свойствах вашего проекта на GitHub, чтобы workflow модуля работал корректно:

- Откройте страницу *Settings -> Actions -> General*.
- Установите параметр *Read and write permissions* в разделе *Workflow permissions*.

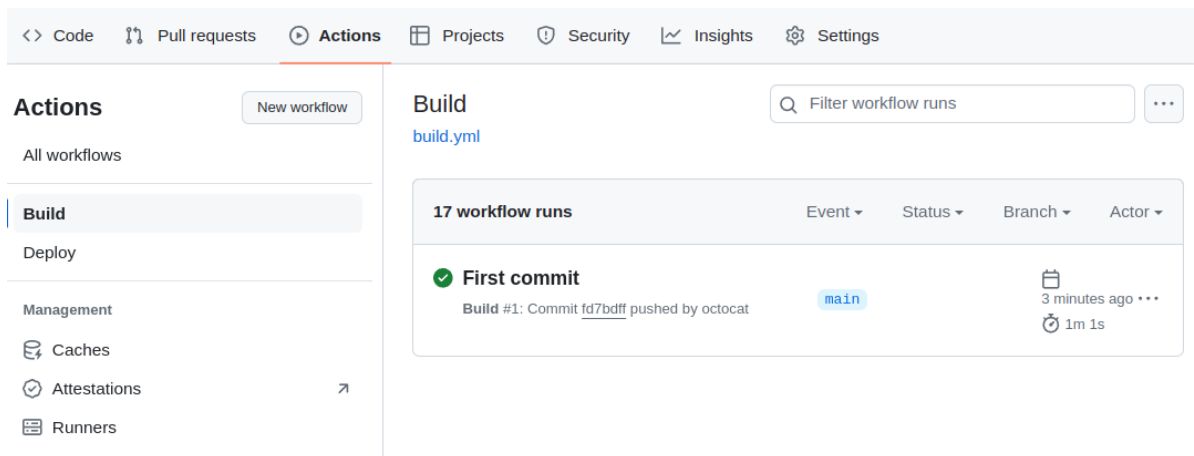
12. Зафиксируйте изменения в репозитории (укажите адрес Git-репозитория модуля).

```
git add .
git commit -m "Initial Commit"
git push --set-upstream origin <GIT_REPO_URL>
```

13. Убедитесь, что сборка модуля выполнена успешно.

Перейдите в раздел *Actions* репозитория модуля и слева, в списке workflow, выберите *Build*. Workflow, запущенный после того, как вы выполнили команду `git push` на предыдущем шаге, должен выполняться успешно.

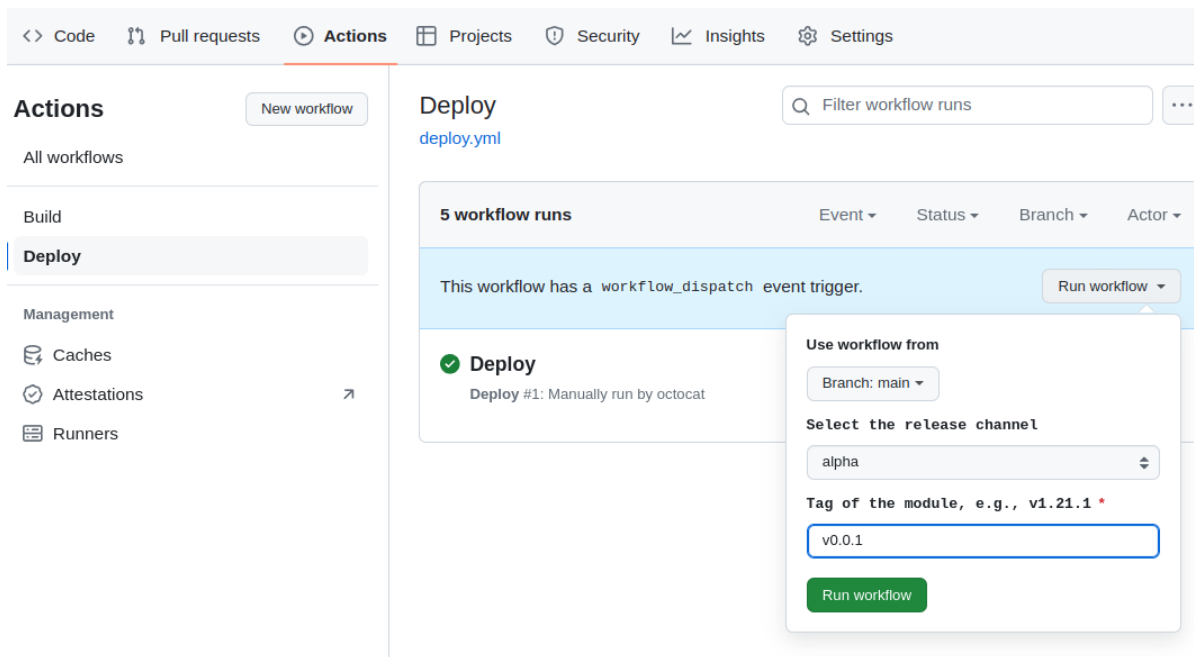
Пример:



Публикация модуля на канале обновлений

Пример публикации версии `v0.0.1` модуля на канале обновлений *Alpha*:

1. Создайте новый релиз модуля `v0.0.1` в репозитории GitHub или установите тег `v0.0.1`.
2. Перейдите в раздел *Actions* репозитория модуля и слева, в списке workflow, выберите *Deploy*.
3. В правой части страницы нажмите на выпадающий список *Run workflow* и выберите `alpha`. Укажите тег `v0.0.1` в поле ввода тега. Нажмите кнопку *Run workflow*.



4. Убедитесь, что workflow публикации модуля выполнен успешно.

Модуль стал доступным для подключения в кластере Deckhouse Kubernetes Platform.

Подключение модуля в кластере

Пример подключения модуля `helloworld` в кластере Deckhouse Kubernetes Platform.

1. Создайте токен доступа в репозитории GitHub с правами для работы с GitHub Packages.

2. Сгенерируйте строку аутентификации для доступа к хранилищу образов контейнеров GitHub Packages в формате `dockerconfigjson`, указав имя пользователя (или организации) GitHub и токен доступа:

```
base64 -w0 <<EOF
{
  "auths": {
    "ghcr.io": {
      "auth": "$(echo -n '<OWNER>:<TOKEN>' | base64 -w0)"
    }
  }
}
EOF
```

3. Создайте в кластере ресурс `ModuleSource` (укажите адрес хранилище образов и строку аутентификации).

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: ghcr
spec:
  registry:
    # Укажите имя пользователя (или организации) GitHub. Например:
    ghcr.io/octocat/modules
    repo: ghcr.io/<!OWNER>/modules
    # Строка аутентификации для доступа к GitHub Packages из предыдущего шага.
    dockerCfg: <!REGISTRY_CREDENTIALS>
EOF
```

Синхронизация данных после создания ресурса может занять несколько секунд.

4. Посмотрите список доступных модулей:

```
$ d8 k get module
NAME          STAGE    SOURCE    PHASE      ENABLED  READY
...
helloworld                    Available  False     False
...
```

5. Создайте ресурс `ModuleUpdatePolicy`, определяющий политику обновления модуля.

Выполните следующую команду, чтобы создать политику обновления с каналом обновления *Alpha* и режимом обновления *Auto*:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: helloworld-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
EOF
```

6. Создайте `ModuleConfig`, где укажите источник модуля (параметр `source`), политику обновления (параметр `updatePolicy`) и установите параметр `enabled` в `true`:

```
d8 k apply -f - <<EOF
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: helloworld
  spec:
    enabled: true
    source: ghcr
    updatePolicy: helloworld-policy
```

7. Проверьте ModuleSource (в статусе не должно содержаться ошибок и должны быть перечислены доступные модули):

```
d8 k get ms ghcr -O yaml
```

8. Убедитесь, что были созданы новые объекты ModuleRelease для модуля:

```
d8 k get mr
```

Пример вывода:

```
$ d8 k get mr
NAME                                PHASE      UPDATE POLICY
TRANSITIONTIME  MESSAGE
helloworld-v0.0.1    Deployed    helloworld-policy    22m
```

9. В случае успешной установки релизов дождитесь перезапуска пода Deckhouse Kubernetes Platform.

```
d8 k -n d8-system get pod -l app=deckhouse
```

Через некоторое время объекты модуля появятся в кластере.

Если при запуске модуля возникли ошибки, посмотрите журнал DKP:

```
d8 k -n d8-system logs deploy/deckhouse -f | jq -rc '.msg'
```

или проверьте состояние очереди DKP:

```
d8 system queue list
```

Миграция ModuleUpdatePolicy на версию v1alpha2

Если в кластере существует ModuleUpdatePolicy версии v1alpha1, то необходимо выполнить следующие шаги по миграции на версию v1alpha2:

Если в кластере в каком-либо ModuleUpdatePolicy версии v1alpha1 определен `moduleReleaseSelector`, то для всех модулей, которые подходят под этот селектор, в системе мониторинга будут гореть алерты `ModuleHasDeprecatedUpdatePolicy`. В этом случае выполните следующие шаги по миграции на версию v1alpha2 ModuleUpdatePolicy:

- Укажите политику обновления для соответствующих модулей в параметре `spec.updatePolicy` ModuleConfig.
- Выполните следующую команду, указав необходимый ModuleUpdatePolicy:

```
d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io <MUP_NAME> --type='json' \
  -p='[{"op": "replace", "path":
"/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}}]'
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Разработка и отладка модуля

При разработке модулей может возникнуть необходимость загрузить и развернуть модуль в обход каналов обновления. Для этого используется ресурс `ModulePullOverride`.



Ресурс `ModulePullOverride` предназначен **только для использования в средах разработки и отладки**. Его применение в production-кластерах **не рекомендуется**. Поддержка ресурса может быть исключена в следующих версиях Deckhouse Kubernetes Platform.

Пример `ModulePullOverride`:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: <module-name>
spec:
  imageTag: <tag of the module image>
  scanInterval: <image digest check interval. Default: 15s>
```

Также для целей разработки может использоваться режим обслуживания модуля (`maintenance mode`). В этом режиме Deckhouse отключает управление ресурсами модуля и не применяет изменения автоматически. Этот режим не предназначен для эксплуатации в production-кластерах.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
...
spec:
  enabled: true
  maintenance: NoResourceReconciliation
  settings:
```

Требования к параметрам ресурса:

- Имя модуля (`metadata.name`) должно соответствовать имени модуля в `ModuleSource` (`.status.modules.[].name`).
- Тег образа контейнера (`spec.imageTag`) может быть любым. Например, `pr333`, `my-branch`.

Необязательный параметр `spec.scanInterval` устанавливает интервал времени для проверки образов в хранилище образов. По умолчанию задан интервал в 15 секунд. Для принудительного обновления можно изменить интервал, либо установить на `ModulePullOverride` аннотацию `renew=""`.

Необязательный параметр `spec.rollback` — если установить этот параметр в `true`, это восстановит развернутый модуль до предыдущего состояния после удаления `ModulePullOverride`.

Результат применения `ModulePullOverride` можно увидеть в сообщении (колонок `MESSAGE`) при получении информации об `ModulePullOverride`. Значение `Ready` означает применение параметров `ModulePullOverride`. Любое другое значение означает конфликт.

Пример отсутствия конфликтов при применении `ModulePullOverride`:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME      UPDATED  MESSAGE  ROLLBACK
example1  10s      Ready    false
```

Требования к модулю:

- Модуль должен существовать, иначе сообщение у `ModulePullOverride` будет *The module not found*.

Пример:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME      UPDATED  MESSAGE                      ROLLBACK
example1  10s      The module not found         false
```

- Модуль не должен быть встроенным модулем Deckhouse, иначе сообщение у `ModulePullOverride` будет *The module is embedded*.

Пример:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME          UPDATED    MESSAGE                      ROLLBACK
ingress-nginx 10s       The module is embedded      false
```

- Модуль должен быть включен, иначе сообщение у ModulePullOverride будет *The module disabled*.

Пример:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME    UPDATED    MESSAGE                      ROLLBACK
example 7s        The module disabled         false
```

- Модуль должен иметь источник, иначе сообщение у ModulePullOverride будет *The module does not have an active source*.

Пример:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME    UPDATED    MESSAGE                      ROLLBACK
example 12s        The module does not have an active source  false
```

- Источник модуля должен существовать, иначе сообщение у ModulePullOverride будет *The source not found*.

Пример:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME    UPDATED    MESSAGE                      ROLLBACK
example 12s        The source not found         false
```

Чтобы обновить модуль не дожидаясь начала следующего цикла обновления, можно выполнить следующую команду:

```
d8 k annotate mpo <name> renew=""
```

Доступность модуля и включение по умолчанию

Чтобы назначить редакции Deckhouse, в которых должен быть доступен модуль, а также наборы модулей (bundles), в составе которых модуль должен быть включен по умолчанию, используйте поле `accessibility` в файле `module.yaml`:

```
name: test
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
```

В данном примере модуль будет доступен в редакции `ee` (DKP Enterprise Edition) и может быть включен через объект `ModuleConfig`, а также будет по умолчанию включен в составе набора модулей `Default`.



- Чтобы использовать этот механизм, файл `module.yaml` должен быть включён в релизный образ.
- Модуль по-прежнему можно отключить с помощью объекта `ModuleConfig`.
- Модуль останется на последнем доступном релизе, если в следующем релизе он будет отключён (например, при установке `available: false` в соответствующей редакции).

Как работает ModulePullOverride

После создания `ModulePullOverride`, соответствующий модуль не будет учитывать `ModuleUpdatePolicy`, а также не будет загружать и создавать объекты `ModuleRelease`. Модуль будет загружаться при

каждом изменении параметра `imageDigest`, после чего будет применяться в кластере. В статусе `ModuleSource` модуль получит признак `overridden: true`, который указывает на то, что используется `ModulePullOverride`, а не `ModuleUpdatePolicy`. Также, соответствующий объект `Module` будет иметь в своем статусе поле `IsOverridden` и версию модуля из `imageTag`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 16
  labels:
    deckhouse.io/epoch: "1326105356"
  name: example
  resourceVersion: "230347744"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
    - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernetes: '> v1.30.0'
  source: example
  version: mpo-tag
  weight: 910
status:
  conditions:
    - lastProbeTime: "2024-12-03T15:57:20Z"
      lastTransitionTime: "2024-12-03T15:57:20Z"
      status: "True"
      type: EnabledByModuleConfig
    - lastProbeTime: "2024-12-03T15:59:58Z"
      lastTransitionTime: "2024-12-03T15:57:26Z"
      status: "True"
      type: EnabledByModuleManager
    - lastProbeTime: "2024-12-03T15:59:58Z"
      lastTransitionTime: "2024-12-03T15:56:23Z"
      status: "True"
      type: IsReady
    - lastProbeTime: "2024-12-03T15:59:48Z"
      lastTransitionTime: "2024-12-03T15:56:47Z"
      status: "True"
      type: IsOverridden
  phase: Ready
```

После удаления `ModulePullOverride` модуль продолжит работать. Но, если для модуля существует `ModuleUpdatePolicy`, то загрузятся новые релизы модуля (`ModuleRelease`), которые заменят текущую “версию разработчика”.

Пример

1. В `ModuleSource` присутствуют два модуля `echo` и `hello-world`. Для них определена политика обновления, они загружаются и устанавливаются в DKP:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
    - name: echo
      policy: test-alpha
    - name: hello-world
      policy: test-alpha
  modulesCount: 2

```

2. Включите модуль и создайте ModulePullOverride для модуля `echo`:

```

apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354

```

После создания ModulePullOverride, для модуля будет использоваться тег образа `registry.example.com/deckhouse/modules/echo:main-patch-03354` (`ms.spec.registry.repo/mpo.metadata.name:mpo.spec.imageTag`).

3. Данные ModulePullOverride будут меняться при каждом обновлении модуля:

```

apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354
  scanInterval: 15s
status:
  imageDigest:
    sha256:ed958cc2156e3cc363f1932ca6ca2c7f8ae1b09ffc1ce1eb4f12478aed1befbc
  message: "Ready"
  updatedAt: "2023-12-07T08:41:21Z"

```

где:

- `imageDigest` — уникальный идентификатор образа контейнера, который был загружен.
- `lastUpdated` — время последней загрузки образа.

4. При этом ModuleSource приобретет вид:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
    - name: echo
      overridden: true
    - name: hello-world
      policy: test-alpha
  modulesCount: 2

```

Логика автообновления модулей

i

Версии ModuleRelease v1.0.0 и v1.1.1 приведены в качестве примера.

1. **Установка модуля.** При включении модуля (`enable module <module name>`) в кластер автоматически загружается и разворачивается актуальная версия модуля из выбранного канала стабильности. Это может быть, например, ModuleRelease v1.0.0. Загружается последняя доступная версия, старые версии не устанавливаются.
2. **Отключение модуля.** При отключении модуля (`disable module <module name>`):
 - Модуль перестаёт получать новые версии.
 - Текущая версия остаётся в кластере в состоянии `Deployed`.
3. **Поведение при повторном включении.**

Если модуль включён в течение 72 часов:

- Используется та же версия, которая была задеплоена ранее (ModuleRelease v1.0.0).
- Проверяются новые релизы.
- При их наличии они загружаются (например, v1.1.0, v1.1.1).
- Далее модуль обновляется в соответствии с обычными правилами обновления (Update). Подробнее.

Если модуль включён позже 72 часов:

- Старая версия удаляется (`delete ModuleRelease v1.0.0`).
- При включении модуля повторно, загружается последняя актуальная версия (например, v1.1.1).
- Начинается тот же цикл, что и при первоначальном включении (см. шаг 1).

4. **Поведение выключенного модуля.** Если модуль отключён, то релизы для него не загружаются. Задеплоенная версия модуля (последняя включённая) удаляется через 72 часа, если модуль так и не был повторно включён.

Пропуск промежуточных релизов (from-to)

Механизм `from-to` позволяет пропускать пошаговые обновления модуля. Если текущая установленная версия модуля (в статусе `Deployed`) попадает в диапазон `from-to`, DKP пропускает промежуточные релизы и устанавливает последнюю доступную версию в пределах `to`.

Чтобы включить механизм, задайте правила перехода в конфигурации модуля (`module.yaml`).
Пример:

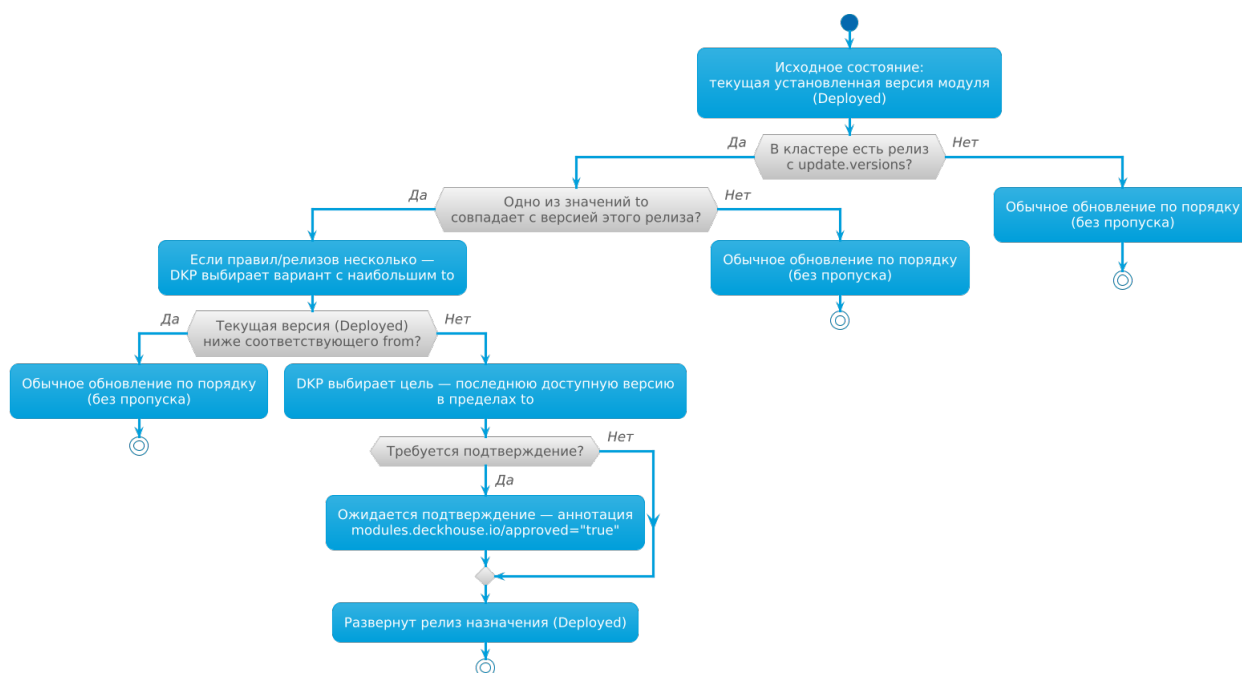
```
update:
  versions:
    - from: "1.67"
      to:   "1.75"
    - from: "1.99"
      to:   "2.0" # Переход между основными версиями задаётся форматом X.0.
```

i Релиз с ограничениями (constrained release) — это релиз модуля, в чьём `module.yaml` задана секция `update.versions`. Механизм `from-to` работает только с такими релизами.

Условия применения `from-to`:

- Версия релиза назначения — правило читается из релиза, на который нужно перейти — значение `to` должно совпадать с версией самого этого релиза (constrained release).
- Текущая установленная версия модуля (Deployed) должна быть не ниже значения `from`. Если текущая версия ниже `from`, переход по правилу не выполняется — обновление идёт по порядку.
- Если одновременно подходят несколько релизов, выбирается вариант с наибольшим `to` (правила могут находиться в разных объектах `ModuleRelease` одного модуля).
- Если ни один релиз не подходит под эти условия, обновление выполняется как обычно — без пропуска промежуточных версий.

Если в кластер попадает релиз с `update.versions`, DKP не требует обновляться по порядку — такой релиз появляется в списке «как есть», DKP автоматически выбирает подходящий вариант и, при необходимости, ждёт подтверждения. Вы можете сразу подтвердить установку последней доступной версии в пределах `to`. После подтверждения промежуточные релизы между `from` и `to` получают статус `Skipped` после реконсильации (не сразу); какое-то время между `Superseded` и `Deployed` возможны релизы в статусе `Pending`.



Проверить доступные релизы (ModuleRelease) можно с помощью команды:

```
d8 k get mr
```

Пример вывода, если текущая версия модуля — `0.3.33`, а в `module.yaml` задано правило `from: "0.3" → to: "0.7"`:

```
p-o-test-v0.1.0    Superseded
p-o-test-v0.2.22   Superseded
p-o-test-v0.3.33   Deployed
p-o-test-v0.4.1    Pending      Release is waiting for the
'modules.deckhouse.io/approved: "true"' annotation
p-o-test-v0.5.27   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.6.11   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.7.25   Pending      Release is waiting for the
'modules.deckhouse.io/approved: "true"' annotation
```

В этом примере вывода текущая установленная версия — 0.3.33 (в статусе Deployed). По правилу from: 0.3 → 0.7 к установке выбирается последняя доступная версия в пределах 0.7 — p-o-test-v0.7.25 (в статусе Pending). После подтверждения версии 0.4.1, 0.5.27 и 0.6.11 становятся в статусе Skipped, а p-o-test-v0.7.25 в статусе Deployed.

Подтвердить релиз модуля можно применив аннотацию с помощью команды:

```
d8 k annotate mr p-o-test-v0.7.25 modules.deckhouse.io/approved="true"
```

Для удобства это можно сделать с помощью d8 CLI (имена модулей и версии автодополняются):

```
d8 system module approve p-o-test v0.7.25
```

Пример вывода после подтверждения:

```
p-o-test-v0.4.1    Skipped
p-o-test-v0.5.27   Skipped
p-o-test-v0.6.11   Skipped
p-o-test-v0.7.25   Deployed
```

Нужна ли аннотация зависит от политики обновления модуля. Подробнее — Политика обновления модуля.

Примеры

Пример 1. Релиз назначения содержит правило, и его версия совпадает с to — переход выполняется.

В этом примере правило from-to прописано в самом релизе назначения, и значение to равно версии этого релиза. Текущая установленная версия (Deployed) не ниже значения from, значит переход возможен. Как только релиз появляется в кластере, DKP готовит обновление. Если по политике обновления требуется подтверждение — пометьте релиз аннотацией. После подтверждения промежуточные релизы будут пропущены, релиз назначения станет Deployed:

```
# Версия модуля (Deployed): v1.67.23 (≥ 1.67) → условие from выполнено.
metadata:
  name: module-test-v1.67.23
status:
  phase: Deployed

# Релиз назначения: версия релиза = update.versions.to (1.75).
metadata:
  name: module-test-v1.75.25
update:
  versions:
    - from: "1.67"
      to: "1.75" # Обновление выполнится до этой версии после подтверждения (если
подтверждение необходимо).
```

Также если текущая версия ещё выше значения from, переход также выполняется:

```
# Версия модуля (Deployed): v1.69.0 (≥ 1.67) → условие from выполнено.
metadata:
  name: module-test-v1.69.0
status:
  phase: Deployed

# Релиз назначения: версия релиза = update.versions.to (1.75)
metadata:
  name: module-test-v1.75.25
update:
  versions:
    - from: "1.67"
      to: "1.75" # Обновление выполнится до этой версии после подтверждения (если
        подтверждение необходимо).
```

Пример 2. Текущая версия ниже `from` — переход не выполняется.

Если текущая установленная версия меньше значения `from`, правило не применяется. Обновление идёт по порядку (без пропуска):

```
# Версия модуля (Deployed): v1.61.0 (< 1.67) → условие from НЕ выполнено.
metadata:
  name: module-test-v1.61.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.75.25
update:
  versions:
    - from: "1.67"
      to: "1.75" # Переход по from-to не выполняется → происходит обновление по порядку.
```

Пример 3. Несовпадение `to` с версией релиза с правилом — переход не выполняется.

Переход возможен только в релиз, где `to` равно версии этого релиза. Если правило описано в другом релизе (например, `to: "1.74"` лежит в релизе `v1.75.25`), оно не сработает — обновление пойдёт по порядку.

```
# Версия модуля (Deployed): v1.67.0.
metadata:
  name: module-test-v1.67.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.74.0 # У этого релиза нет update.versions.

# Правило записано в другом релизе (v1.75.25), и to не равно его версии.
metadata:
  name: module-test-v1.75.25
update:
  versions:
    - from: "1.67"
      to: "1.74" # Значение to не равно версии этого релиза (1.75.25) → правило
        игнорируется.
```

Артефакты модуля в хранилище образов

После сборки модуля его артефакты должны быть загружены в хранилище образов контейнеров по пути, который является *источником* для загрузки и запуска модулей в DKP. Путь, по которому загружаются артефакты модулей в хранилище образов, указывается в ресурсе `ModuleSource`.

Пример иерархии образов контейнеров после загрузки артефактов модулей `module-1` и `modules-2` в registry:

```

registry.example.io
├─ modules-source
│ └─ module-1
│   ├── v1.23.1
│   ├── d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
│   ├── e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
│   ├── v1.23.2
│   └─ release
│     ├── v1.23.1
│     ├── v1.23.2
│     ├── alpha
│     └─ beta
└─ module-2
  ├── v0.30.147
  ├── d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
  ├── e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
  ├── v0.31.1
  └─ release
    ├── v0.30.147
    ├── v0.31.1
    ├── alpha
    └─ beta

```



Хранилище образов должно поддерживать вложенную структуру репозитория. Подробнее об этом в разделе требований.

Далее приведен список команд для работы с источником модулей. В примерах используется утилита `crane`. Установите ее по инструкции. Для macOS воспользуйтесь `brew`.

Вывод списка модулей в источнике модулей

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>
```

Пример:

```

$ crane ls registry.example.io/modules-source
module-1
module-2

```

Вывод списка образов модуля

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>
```

Пример:

```

$ crane ls registry.example.io/modules-source/module-1
v1.23.1
d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
v1.23.2

```

В примере в модуле `module-1` присутствуют два образа модуля и два образа контейнеров приложений.

Вывод файлов в образе модуля

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -tf -
```

Пример:

```
crane export registry.example.io/modules-source/module-1:v1.23.1 - | tar -tf -
```

Ответ будет достаточно большим.

Вывод списка образов контейнеров приложений модуля

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -0xf - images_digests.json
```

Пример:

```
$ crane export registry.example.io/modules-source/module-1:v1.23.1 - | tar -0xf - images_digests.json
{
  "backend": "sha256: fcb04a7fed2c2f8def941e34c0094f4f6973ea6012ccfe2deadb9a1032c1e4fb",
  "frontend": "sha256: f31f4b7da5faa5e320d3aad809563c6f5fcaa97b571fffa5c9cab103327cc0e8"
}
```

Конфигурация дополнительных образов

Модули могут включать дополнительные образы (например, базы данных уязвимостей или другие вспомогательные данные) путем добавления файла `extra_images.json`. Этот файл указывает дополнительные образы, которые необходимо вручную загрузить в реестр и которые отделены от основных образов модуля.

Для просмотра конфигурации дополнительных образов:

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -0xf - extra_images.json
```

Пример файла `extra_images.json` для базы уязвимостей `neuvector`:

```
{
  "scanner": 3
}
```

Важные замечания:

- Дополнительные образы должны быть вручную загружены в реестр модулей по пути `extra/`.
- Используйте команду `d8 mirror pull --only-extra-images` для загрузки только дополнительных образов.
- Дополнительные образы хранятся в реестре как `<имя-модуля>/extra/<имя-образа>`.

Просмотр списка релизов

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
```

Пример:

```
$ crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
v1.23.1
v1.23.2
alpha
beta
```

В примере в хранилище образов два релиза и используются два канала обновлений: `alpha` и `beta`.

Вывод версии, используемой на канале обновлений `alpha`

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release:alpha - | tar -0xf - version.json
```

Пример:

```
$ crane export registry.example.io/modules-source/module-1/release:alpha - | tar -0xf - version.json
{"version": "v1.23.2"}
```

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `deckhouse`

Катастрофоустойчивость

Катастрофоустойчивость — способность инфраструктуры на базе Deckhouse Kubernetes Platform

(DKP) сохранять работоспособность при масштабных отказах. Она обеспечивается за счёт распределённого развёртывания, автоматического переключения трафика и репликации критически важных компонентов.

В DKP реализованы два подхода к обеспечению катастрофоустойчивости:

- Геораспределённость — распределение элементов инфраструктуры по разным зонам доступности (Multi-AZ) или регионам (Multi-Region). Такой подход позволяет снизить влияние отказов инфраструктуры на работу приложений. Подробнее — в разделе Геораспределённость.
- Георезервирование основано на использовании нескольких независимых кластеров, объединённых в мультикластерную систему. При недоступности одного из кластеров трафик может быть автоматически перенаправлен в другой. Подробнее — в разделе Георезервирование.

Оба подхода требуют настройки сетевой связности между узлами и регионами, балансировки трафика и правильной организации хранилищ. Выбор конкретной архитектуры зависит от особенностей приложений и ограничений инфраструктуры.

Геораспределённость

Геораспределённость в контексте катастрофоустойчивости Deckhouse Kubernetes Platform (DKP) — архитектурный подход, обеспечивающий непрерывную работу кластера за счёт распределения его компонентов по нескольким зонам доступности (Multi-AZ) или регионам (Multi-Region). Такой подход минимизирует риски аварий различных масштабов.



При реализации геораспределённости важно обеспечить сетевую связанность между узлами кластера. Эту задачу решает администратор платформы.

Использование нескольких зон доступности (Multi-AZ)

DKP предоставляет возможность распределения узлов кластера по зонам доступности. Это повышает устойчивость приложений к сбоям на уровне инфраструктуры и обеспечивает их работоспособность даже при недоступности одной из зон в ЦОДе или облаке.

Зона доступности (Availability Zone, AZ) — базовая единица катастрофоустойчивой архитектуры. Она представляет собой логическую группу из одного или нескольких ЦОДов, расположенных на расстоянии нескольких километров друг от друга. AZ можно рассматривать как изолированную единицу (виртуальный дата-центр), хотя физически это распределённая инфраструктура. Зоны доступности входят в состав регионов.

Характеристики зон доступности

Зоны доступности обладают следующими характеристиками:

- разнесение ЦОДов внутри одной AZ на достаточное расстояние (обычно ≤ 10 км) для защиты от локальных инцидентов (наводнение, пожар) при сохранении низкой сетевой задержки;
- независимое электроснабжение для компонентов, образующих зону доступности (разные электрические подстанции, дизель-генераторы и т.д.);
- использование компонентами зоны разных провайдеров и маршрутов;
- относительно низкая задержка внутри зоны.

Распределение узлов по зонам

В большинстве облачных провайдеров распределение по зонам происходит автоматически. При работе вне облаков администратору необходимо вручную маркировать узлы с помощью лейблов, указывающих зону.

Процесс маркировки узлов лейблами и их добавления в группы описан в разделе «Управление узлами».

Использование нескольких регионов (Multi-Region)

DKP поддерживает распределение узлов кластера по разным регионам. Это даёт возможность приложениям продолжать работу даже при выходе из строя целого ЦОДа или облака.

Регион (Region) — независимая географическая область, включающая несколько (обычно не менее трёх) изолированных зон доступности (AZ). Все зоны региона объединены в единую логическую инфраструктурную единицу с общими сервисами управления. Регион — ключевой элемент глобальной распределённой архитектуры облачных провайдеров.

i Использование нескольких регионов (Multi-Region) обычно подходит только для ситуаций, когда приложения могут функционировать в каждом регионе автономно (не обращаясь к другим регионам). Это обусловлено относительно высокой задержкой (Latency) между регионами. Например, размещение кеширующих серверов для CDN или узлов распределённого мониторинга.

Характеристики регионов

Регионы обладают следующими характеристиками:

- Разница в задержке внутри региона и между регионами. Если внутри региона задержка при передаче данных от AZ до AZ может составлять около 5 мс, то между регионами — от 50 до 200 мс.
- Географическая удалённость. Регионы могут быть существенно удалены друг от друга (например, расстояние между `europa-west3` (Франкфурт) и `us-east-1` (Вирджиния) — ~6500 км).
- Физическая безопасность. Провайдеры стараются размещать регионы вне зон сейсмической или политической нестабильности.
- Энергетическая избыточность. Регионы, как правило, подключаются к нескольким национальным энергосетям (например, `me-central-1` (ОАЭ) использует 4 независимых источника).

Распределение узлов по разным регионам

Для распределения узлов по регионам необходимо:

- маркировать каждый узел лейблом, указывающим регион;
- обеспечить стабильные каналы связи между регионами.

Процесс маркировки узлов лейблами и их добавления в группы узлов описан в разделе «Управление узлами».

Балансировка входящего трафика

В геораспределённом кластере балансировка трафика может быть организована средствами облачного провайдера, а также средствами внутреннего балансировщика (например, MetalLB). Конкретная реализация зависит от инфраструктуры и настраивается администратором. Дополнительные сведения о вариантах балансировки и их настройках — в разделе Сеть.

Организация хранилища

Организацию и настройку систем хранения данных в геораспределённом кластере выполняет администратор. DKP поддерживает различные варианты хранилищ, выбор которых зависит от требований конкретного проекта. Подробнее о системах хранения, их особенностях и настройке — в разделе Хранение.

Георезервирование

Георезервирование в контексте катастрофоустойчивости Deckhouse Kubernetes Platform (DKP) — метод повышения отказоустойчивости, при котором несколько независимых кластеров Kubernetes объединяются в единую мультикластерную систему (мультикластер), распределённую по разным географическим локациям.

DKP предлагает разные инструменты для объединения нескольких кластеров в мультикластер. Это обеспечивает автоматическую маршрутизацию трафика (внешнего и внутреннего) в другой кластер с доступной копией приложения, если оно недоступно в исходном.

Кластеры можно объединить:

- с помощью встроенного Service Mesh на основе Istio,
- с использованием сетевого функционала DKP на базе Cilium.

Для включения функции мультикластер необходимо развернуть по одному кластеру с DKP в каждом регионе, а затем объединить их используя декларативный API. Важно, чтобы приложение могло работать параллельно в разных регионах, а также чтобы между регионами существовал стабильный сетевой канал. Правильно настроенная система обеспечивает глобальную катастрофоустойчивость приложения, гарантируя его бесперебойную работу даже при выходе из строя целого региона или облака.

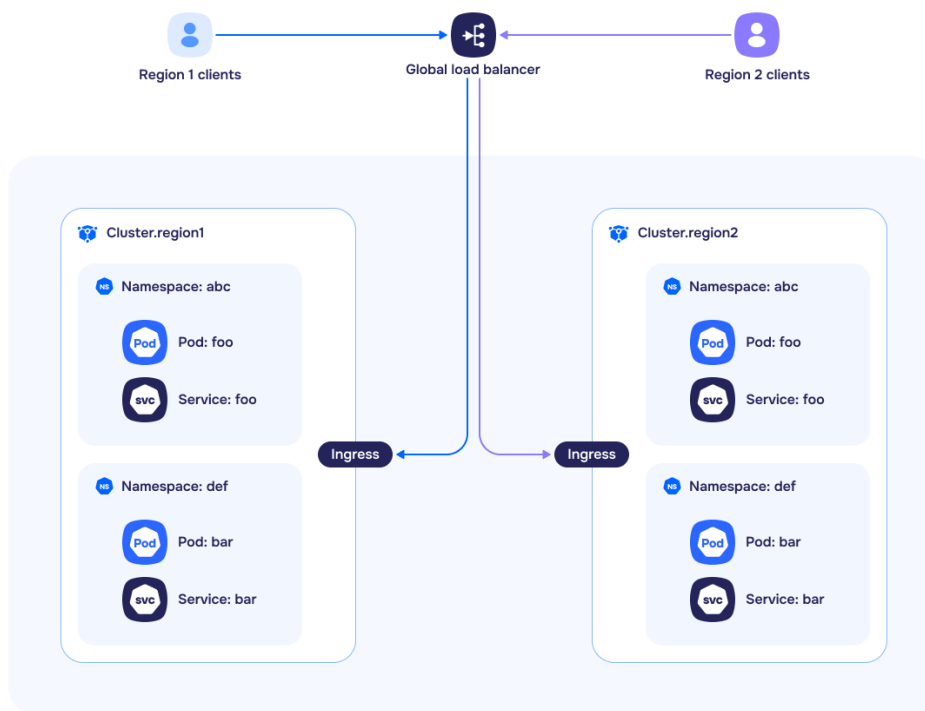
Процесс объединения нескольких кластеров в мультикластер описан в разделе «Межкластерное взаимодействие».

Балансировка входящего трафика в мультикластере

Для обеспечения катастрофоустойчивости в мультикластере может быть реализована балансировка входящего трафика внешним балансировщиком по одной из схем: `active-active` или `active-standby`. Балансировка настраивается администратором кластера.

Балансировка по схеме `active-active`

При использовании схемы `active-active` внешний балансировщик распределяет трафик между несколькими активными кластерами, например, по географическому признаку или на основе текущей загрузки.

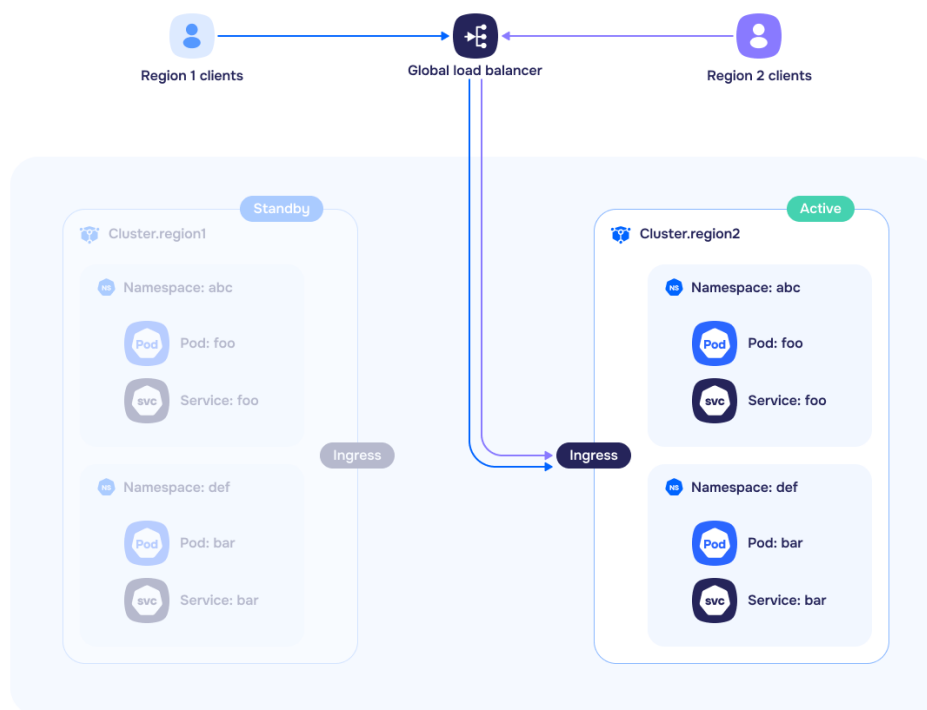


Особенности балансировки по схеме `active-active`:

- Все кластеры в мультикластере обрабатывают трафик одновременно.
- Трафик направляется в ближайший к клиенту, наименее загруженный кластер или используются иные правила.
- Общие данные реплицируются между кластерами.
- При сбое одного кластера трафик переключается на работающие (автоматический failover).

Балансировка по схеме `active-standby`

При использовании схемы `active-standby` внешний балансировщик направляет трафик в кластер, который активен в данный момент. Неактивный кластер является резервным, трафик в него будет направлен, если основной кластер будет недоступен.



Особенности балансировки по схеме `active-standby`:

- Активный кластер принимает весь трафик.
- Резервный кластер бездействует до момента сбоя основного.
- Переключение трафика с активного на резервный кластер (failover) происходит автоматически при обнаружении проблем.

Организация хранилища в мультикластере

За организацию и настройку хранилища в мультикластере отвечает администратор. В DKP поддерживаются разные системы хранения (хранилища). Выбор подходящего варианта зависит от требований к производительности, отказоустойчивости и способам синхронизации данных. Подробнее о системах хранения, их особенностях и настройке — в разделе [Хранение](#).

Обновление

Релизы

Релизом считается любая опубликованная версия платформы Deckhouse Kubernetes Platform (DKP). Каждый релиз распространяется по каналам обновлений с указанными задержками. DKP публикует два вида релизов:

- **патч-версия** (например, с `0.0.1` на `0.0.2`) — исправление ошибок, выпускается по мере необходимости;
- **минорная версия** (например, с `0.0.1` на `0.1.0`) — добавление новой функции, выпускается каждые 3-4 недели.

Каналы обновлений



Актуальная информация о версиях DKP на разных каналах обновлений доступна на сайте releases.deckhouse.ru.

Deckhouse Kubernetes Platform использует **пять каналов обновлений**, обеспечивающих поэтапное развертывание новых версий. Каждая новая версия DKP сначала публикуется в канале **Alpha** и постепенно продвигается к **Rock Solid**. Обновления из менее стабильных каналов доступны небольшому числу пользователей, что позволяет выявлять и устранять потенциальные проблемы до того, как они повлияют на production-среды.

- **Rock Solid** — наиболее стабильный канал обновлений. Подойдет для кластеров, которым необходимо обеспечить повышенный уровень стабильности. Обновления публикуются не ранее чем через 30 дней после выхода релиза.
- **Stable** — стабильный канал для кластеров в эксплуатации. Обновления публикуются не ранее чем через 14 дней после выхода релиза.
- **Early Access** — рекомендуемый канал обновлений, если вы не уверены в выборе. Подойдет для активно развивающихся кластеров (запускаются, дорабатываются новые приложения и т. п.). Обновления публикуются не ранее чем через 7 дней после выхода релиза.
- **Beta** — ориентирован на кластеры разработки, как и канал обновлений Alpha. Получает версии, предварительно опробованные на канале обновлений Alpha.
- **Alpha** — наименее стабильный канал обновлений с наиболее частым появлением новых версий. Ориентирован на кластеры разработки с небольшим количеством разработчиков. Версии появляются сразу после выхода релиза.

Процесс переключения на более стабильный канал

При переключении на более стабильный канал (например, с `Alpha` на `EarlyAccess`):

1. DKP загружает данные о релизах из канала `EarlyAccess`.
2. Сравнивает их с данными из существующих в кластере кастомных ресурсов `DeckhouseRelease`.
 - Если в кластере есть более поздние релизы со статусом `Pending` (ещё не применены), они будут **удалены**, поскольку они ещё не были опубликованы на новом канале.
 - Если более поздние релизы уже перешли в статус `Deployed` (успешно установлены), переход на новый релиз произойдёт не сразу. DKP останется на текущем релизе до тех пор, пока на канале `EarlyAccess` не выйдет более поздняя версия, которую можно будет установить.

Процесс переключения на менее стабильный канал

При переключении на менее стабильный канал (например, с `EarlyAccess` на `Alpha`):

1. DKP скачивает данные о релизах из канала `Alpha`.
2. Сравнивает их с данными из существующих в кластере кастомных ресурсов `DeckhouseRelease`.
3. Выполняет обновление в соответствии с настроенными параметрами обновления.

Обновление control plane

В Deckhouse Kubernetes Platform (DKP) процесс обновления control plane максимально автоматизирован и безопасен как для single-master-, так и для multi-master-кластеров. При этом могут возникать кратковременные перерывы в доступности API-сервера, однако на работу приложений внутри кластера обновление control plane не влияет. Дополнительное окно для регламентных работ, как правило, не требуется.

В DKP поддерживаются последние пять минорных версий Kubernetes. Полный список актуальных версий доступен в соответствующей таблице.

Обновление патч-версий

Обновление patch-версий компонентов control plane (в рамках одной минорной версии, например, с 1.27.3 на 1.27.5) происходит автоматически вместе с обновлением версии Deckhouse. Пользователь не может управлять установкой patch-версий вручную — процесс полностью автоматизирован внутри платформы.

Автоматическое обновление минорных версий

Для автоматического обновления минорной версии control plane (например, с 1.28.* на 1.30.*) укажите `kubernetesVersion: Automatic` в ресурсе `ClusterConfiguration`. Будет выбрана минорная версия Kubernetes, используемая в DKP по умолчанию на момент обновления.

Ручное обновление минорных версий

Для ручного обновления минорной версии control plane (например, с 1.28.* до 1.30.*), укажите нужную версию в параметре `kubernetesVersion` ресурса `ClusterConfiguration`. Например: `kubernetesVersion: 1.30`.

```
d8 system edit cluster-configuration
```

Команда запускает обновление до минорной версии Kubernetes, используемой в DKP по умолчанию на момент обновления. Чтобы отследить ход обновления, ориентируйтесь на версию Kubernetes, указанную в выводе команды описания узлов:

```
d8 k get nodes
```

Процесс обновления компонентов control plane

1. Модуль control-plane-manager вносит изменения в манифесты основных компонентов (apiserver, controller-manager, scheduler и др.).
2. Обновлённые компоненты разворачиваются на всех master-узлах.

При обновлении до новой минорной версии (**upgrade**):

- Если нужно поднять версию сразу на несколько минорных релизов (например, с 1.28 до 1.30), обновление происходит поэтапно: с 1.28 до 1.29 и затем с 1.29 до 1.30.
- На каждом этапе сначала обновляется control plane, а затем выполняется обновление kubelet на узлах.

При откате до предыдущей версии (**downgrade**, не поддерживается для редакций CSE):

- Гарантируется успешный откат только на одну минорную версию назад от максимальной версии, когда-либо использовавшейся в кластере.
- Откат происходит в обратном порядке. Сначала откатываются компоненты kubelet на узлах, затем — компоненты control plane.

После обновления control plane начинается обновление узлов кластера:

1. Bashible API Server меняет версию kubelet в скриптах для всех групп узлов (NodeGroup).
2. Обновление узлов в разных NodeGroup идёт параллельно, однако внутри каждой NodeGroup узлы обновляются последовательно, по одному.
3. Для каждой NodeGroup учитываются её настройки:
 - ручной или автоматический режим обновления;
 - наличие окон обновлений;
 - требуется ли удалять поды с узла перед обновлением и др.
4. Выбирается один (или несколько) узлов-кандидатов, удовлетворяющих всем условиям обновления, после чего выполняется их обновление.
5. Процесс повторяется, пока не обновятся все узлы во всех группах.
6. Когда обновлены все узлы во всех группах, процесс обновления завершается.

Получение истории изменений (changelog)

При каждом выпуске новой версии Deckhouse Kubernetes Platform (DKP) формируется *changelog* — подробный список изменений, включающий новые возможности, исправления ошибок, обновления компонентов, а также важные замечания по совместимости.

Changelog для конкретной версии DKP можно найти в общем списке релизов проекта Deckhouse на GitHub.

Сводная информация о ключевых изменениях, обновлениях версий компонентов, а также о том, какие компоненты в кластере будут перезапущены в процессе обновления, доступна в описании нулевой патч-версии релиза: пример для релиза DKP v1.68.

Содержимое changelog

Changelog содержит четыре раздела:

- **Know before update (важно знать перед обновлением)** — важная информация, которую необходимо учесть перед обновлением. Включает сведения о требованиях к совместимости, необходимых действиях перед обновлением и возможных последствиях для кластера.
- **Features (новые возможности)** — описание ключевых улучшений и новых функций, добавленных в релизе.
- **Fixes (исправления)** — описание незначительных изменений, обновлений безопасности или улучшения производительности.
- **Chore (сервисные изменения)** — перечисление технических изменений, которые включают обновления зависимостей, рефакторинг, исправления в процессах сборки и тестирования, а также устранение уязвимостей.

Минорные версии и нулевая патч-версия

Вся ключевая информация о крупных изменениях добавляется в **нулевую патч-версию** релиза (например, v1.68.0 для релиза v1.68). Перед обновлением до новой минорной версии (например, v1.68):

1. Изучите changelog соответствующей версии.
2. Проверьте, затрагивают ли изменения вашу инфраструктуру.
3. При необходимости внесите корректировки в конфигурацию кластера.

Проверка зависимостей при обновлении

Перед применением нового релиза Deckhouse Kubernetes Platform проверяет состояние кластера. При обнаружении одной из следующих несовместимостей, процесс обновления прерывается:

- Неподдерживаемая версия Kubernetes.
- При включённом режиме `kubernetesVersion: Automatic` обновление останавливается, если:
 - в релизе изменена рекомендуемая версия Kubernetes;
 - включен мониторинг;
 - в кластере присутствуют ресурсы, объявленные как `deprecated` в новой версии.
- Установленная версия Ingress-контроллера не поддерживается новым релизом.
- На узлах присутствуют устаревшие или неподдерживаемые операционные системы.
- В кластере включён модуль, объявленный `deprecated` или удалённый в новом релизе.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `control-plane-manager`

Подсистема Deckhouse

В данном подразделе описана архитектура модулей, входящих в подсистему Deckhouse платформы Deckhouse Kubernetes Platform (DKP).

В подсистему Deckhouse входят следующие модули:

- `deckhouse` — контроллер Deckhouse;
- `console` — веб-интерфейс Deckhouse;
- `deckhouse-tools` — создает веб-интерфейс для скачивания CLI-утилиты `dh`;
- `documentation` — создает веб-интерфейс с документацией, соответствующей запущенной версии DKP;
- `registry` — управляет конфигурацией компонентов DKP, отвечающих за работу с хранилищем образов контейнеров, и предоставляет внутреннее хранилище образов.

Модули веб-интерфейса

Модуль `console`

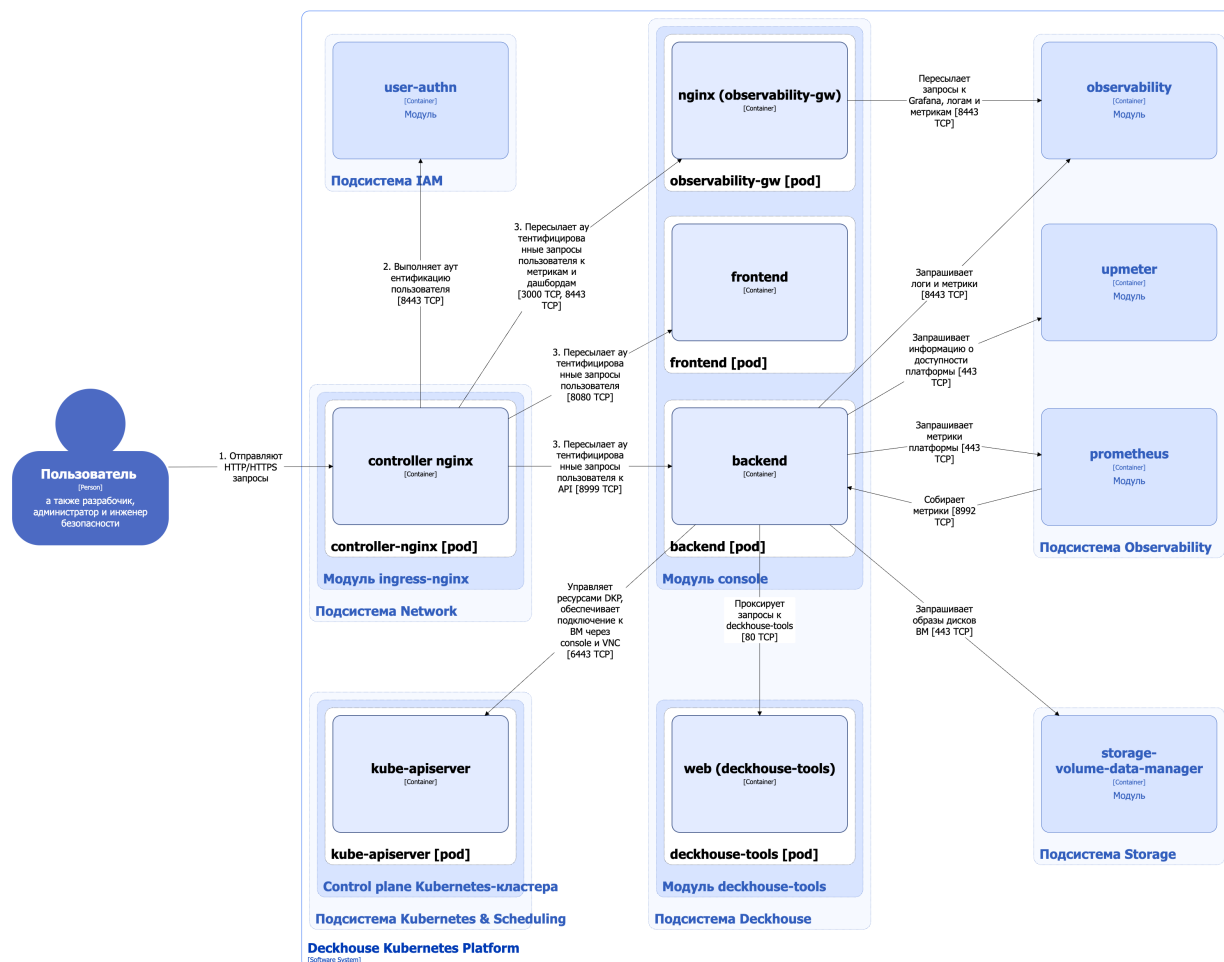
Модуль `console` реализует веб-интерфейс Deckhouse Kubernetes Platform (DKP), который упрощает управление платформой и позволяет отслеживать состояние системы.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `console` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующей диаграмме:



i

Номерами на схеме отмечен порядок прохождения запроса пользователя к компонентам `frontend`, `backend` и `nginx`.

На шагах 1, 2 и 3 запрос проходит через Ingress NGINX Controller, где выполняется обязательная аутентификация пользователя с использованием модуля `user-authn`. Подробнее об архитектуре модуля `user-authn` можно узнать в соответствующем разделе документации.

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Frontend** — состоит из одного контейнера `frontend` и предоставляет веб-интерфейс для пользователей и администраторов DKP.
2. **Backend** — состоит из одного контейнера `backend` и реализует API-интерфейс, обеспечивающий следующие возможности:
 - получение, создание, удаление и изменение ресурсов DKP в соответствии с правами пользователя;
 - генерация `kubeconfig` с профилем пользователя;
 - определение окружения, в котором развернут кластер DKP;
 - обнаружение версий DKP и Kubernetes;
 - загрузка метрик и логов;
 - загрузка информации о доступности платформы;
 - скачивание дисков VM (при включенных модулях `virtualization` и `storage-volume-data-manager`).
3. **Observability-gw** — состоит из одного контейнера `nginx` и выполняет проксирование запросов к Grafana для встраивания дашбордов в основной веб-интерфейс платформы, а также для работы с метриками и логами выбранного проекта.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - организация подключения к VM через `console` и `VNC`;

- создание, удаление, изменение и отслеживание ресурсов DKP.
- 2. **Uptimeer** — получение информации о доступности DKP.
- 3. **Deckhouse-tools** — пересылка запросов для загрузки утилиты Deckhouse CLI.
- 4. **Prometheus** — получение системных метрик (CPU, RAM и др.) платформы.
- 5. **Observability** — получение метрик и логов для выбранного проекта.
- 6. **Storage-volume-data-manager** — экспорт образов дисков VM.

С модулем взаимодействуют следующие внешние компоненты:

- **Controller nginx** — пересылка внешних запросов пользователя к веб-интерфейсу модуля.

Модуль documentation

Модуль `documentation` — это реализация веб-интерфейса с документацией, соответствующей запущенной версии DKP.

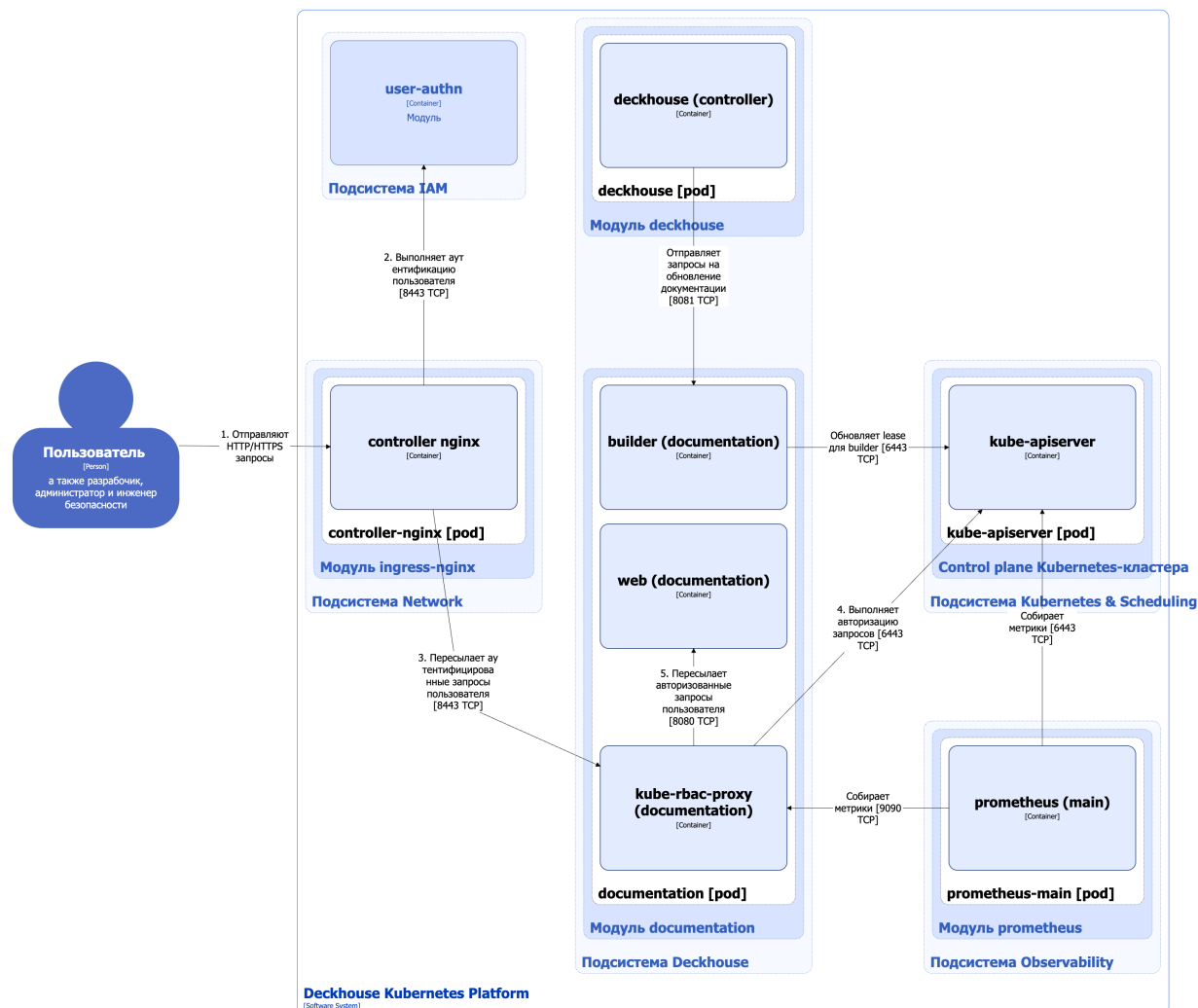
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `documentation` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующей диаграмме:



- i** Номерами на схеме отмечен порядок прохождения запроса пользователя к компоненту `web`:
- на шагах 1, 2 и 3 запрос проходит через Ingress NGINX Controller, где выполняется обязательная аутентификация пользователя с использованием модуля `user-authn`;
 - на шагах 4 и 5 выполняется авторизация пользователя на основе Kubernetes RBAC для организации защищенного доступа.

Компоненты модуля

Модуль состоит из одного компонента:

- **Documentation** — компонент, реализующий веб-интерфейс документации.

Состоит из следующих контейнеров:

- **web** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к основному контейнеру. Является Open Source-проектом;
- **builder** — сайдкар-контейнер, который динамически расширяет документацию при установке новых модулей DKP. Для рендеринга и генерации актуального содержимого сайта используется генератор статических сайтов Hugo.

Контейнер **builder** автоматически создаёт и обновляет Kubernetes-ресурс Lease, размещая в нём эндпоинт для обновления документации. Этот эндпоинт используется контроллером модуля `deckhouse` для запуска обновления документации при обновлении или установке модулей. Благодаря этому изменения своевременно отображаются в веб-интерфейсе.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

- **Kube-apiserver**:
 - создание и обновление ресурса Lease;
 - авторизация запросов к веб-интерфейсу документации.

С модулем взаимодействуют следующие внешние компоненты:

1. **Deckhouse** — отправка запросов на обновление документации при изменении состава модулей.
2. **Prometheus** — сбор метрик модуля.
3. **Controller nginx** — пересылка внешних запросов пользователя к веб-интерфейсу модуля.

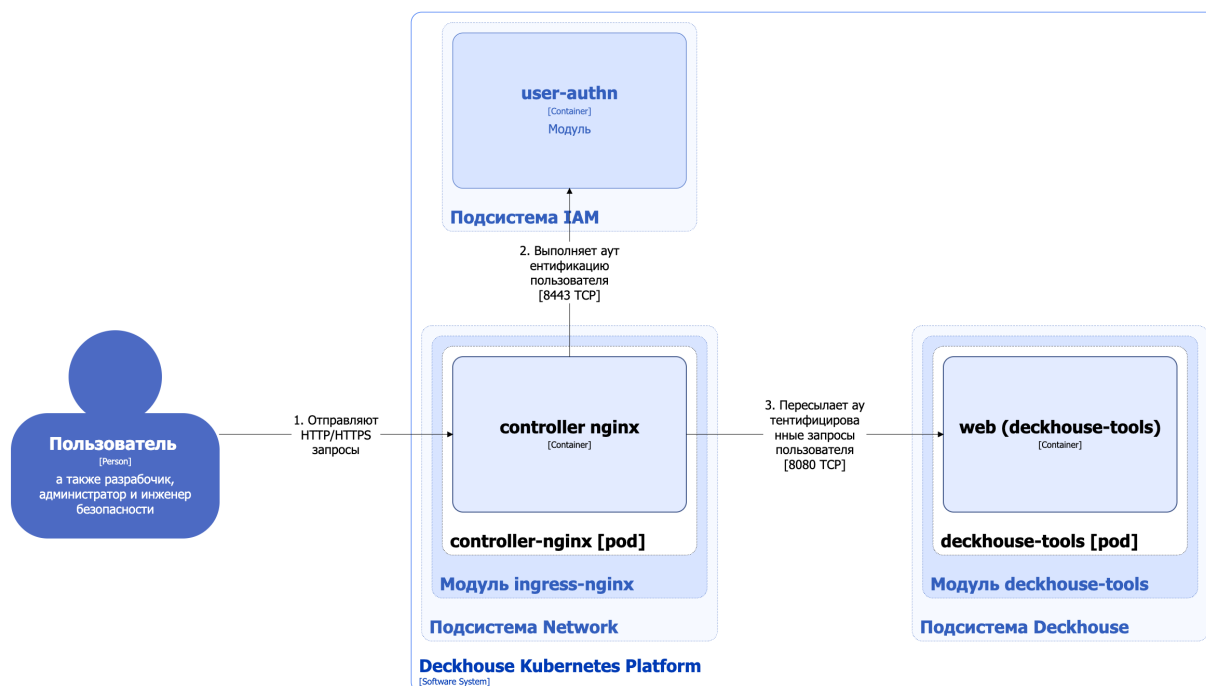
Модуль `deckhouse-tools`

Модуль `deckhouse-tools` — это реализация веб-интерфейса со ссылками на скачивание утилиты Deckhouse CLI под различные операционные системы.

Архитектура модуля

- i** Для упрощения схемы приняты следующие допущения:
- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
 - Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `deckhouse-tools` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующей диаграмме:



i

Номерами на схеме отмечен порядок прохождения запроса пользователя к компоненту `web`.

На шагах 1, 2 и 3 запрос проходит через Ingress NGINX Controller, где выполняется обязательная аутентификация пользователя с использованием модуля `user-authn`.

Компоненты модуля

Модуль состоит из одного компонента:

- **Deckhouse-tools** — состоит из одного контейнера `web` и реализует веб-интерфейс со ссылками на скачивание утилиты Deckhouse CLI.

Взаимодействия модуля

С модулем взаимодействуют следующие внешние компоненты:

- **Controller nginx** — пересылка внешних запросов пользователя к веб-интерфейсу модуля.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `deckhouse-tools`
- Документация модуля `prometheus`
- Документация модуля `storage-volume-data-manager`
- Документация модуля `upmeter`
- Документация модуля `virtualization`

Модуль deckhouse

Модуль `deckhouse` реализует ядро Deckhouse Kubernetes Platform (DKP) и выполняет следующие операции:

- обновление платформы;
- управление конфигурацией модулей;
- установка и обновление модулей;
- запуск сборки документации модулей;
- валидация кастомных ресурсов, находящихся под управлением модулей DKP.

Модуль управляет следующими кастомными ресурсами API-группы `deckhouse.io`:

- управление модулями:
 - `Module` — описание, статус и публикация информации о модуле;
 - `ModuleConfig` — описание пользовательских настроек для модулей;
 - `ModulePullOverride` — описание исключений для выбора версий модулей;
 - `ModuleRelease` — описание, публикация и отслеживание релизов модулей;
 - `ModuleSettingsDefinition` — схема, версии и правила преобразования настроек модуля;

- ModuleSource — описание источника, репозитория или хранилища модулей;
 - ModuleUpdatePolicy — правила обновления и автоматизации переходов версий модулей;
- управление платформой:
 - DeckhouseRelease — объект, определяющий релиз (версию) DKP и политику обновления платформы;
- управление пакетами (Marketplace):
 - Application — описание и желаемое состояние прикладного пакета (группы компонентов или приложения);
 - ApplicationPackage — метаданные, источники и настройки пакета;
 - ApplicationPackageVersion — описание конкретной версией пакета и ее параметров;
 - PackageRepository — объект, описывающий источник репозитория пакетов и их параметры;
 - PackageRepositoryOperation — операции над репозиториями пакетов, такие как синхронизация или обновление;
- управление утилитами:
 - CNIMigration — процесс миграции сетевого плагина Container Network Interface (CNI), содержит параметры и статус миграции;
 - CNINodeMigration — статус и управление миграцией CNI на уровне отдельных узлов;
 - ObjectKeeper — ресурс, обеспечивающий связь между другими ресурсами Kubernetes с использованием ownerReference;
 - ModuleDocumentation — описание параметров для генерации и хранения документации модулей;
- управление кастомными ресурсами под управлением модулей DKP:
 - ConversionWebhook — настройки и обработчики вебхуков для конверсий ресурсов;
 - ValidationWebhook — настройки и обработчики вебхуков для валидации ресурсов.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `deckhouse` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Deckhouse** (Deployment) — контроллер, реализующий операции по управлению платформой.

Контроллер координирует задачи по управлению платформой с использованием механизма очередей.

Контроллер Deckhouse может быть запущен в стандартном режиме или в режиме изоляции хуков. Для этого необходимо создать ConfigMap `chroot-mode` в неймспейсе `d8-system`. В режиме изоляции `shell-хуки` и скрипты включения модулей выполняются в `chroot-окружении` с ограниченным набором смонтированных каталогов, что изолирует их от файловой системы контейнера контроллера.

Если включен режим высокой доступности (High Availability, HA), запускается несколько экземпляров контроллера Deckhouse. Для обеспечения корректной работы контроллеры Deckhouse проводят выборы лидера с использованием ресурса `Lease deckhouse-leader-election`. Контроллер, который был избран как лидер, берёт на себя выполнение всех операций по управлению платформой.

Кроме того, с помощью контроллера Deckhouse можно настроить следующие параметры:

Параметр в конфигурации модуля	Описание
<code>logLevel</code>	Уровень логирования
<code>bundle</code>	Набор модулей, включенных по умолчанию
<code>releaseChannel</code>	Канал обновлений
<code>update.mode</code>	Режим обновлений

Параметр в конфигурации модуля	Описание
<code>update.windows.days</code>	Окна обновлений

Подробнее с описанием настроек модуля можно ознакомиться в разделе документации модуля.

Состоит из следующих контейнеров:

- **init-downloaded-modules** — init-контейнер, подготавливающий структуру каталогов для работы с модулями;
 - **deckhouse** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к интерфейсу отладки компонентов основного контейнера.
- 2. **Webhook-handler** (Deployment) — состоит из одного контейнера **handler** и реализует универсальный вебхук для конверсий и валидации кастомных ресурсов, находящихся под управлением DKP.

Компонент следит за кастомными ресурсами `ConversionWebhook` и `ValidationWebhook` и на их основе создаёт из шаблона Python-файлы хуков для `shell-operator`. При получении запросов от `kube-apiserver` на валидацию или конверсию ресурсов `shell-operator` запускает необходимый хук и возвращает результат обработки.
- 3. **Cni-migration-manager** (Deployment) — опциональный компонент, запускающийся на узлах control plane и состоящий из одного контейнера **manager**. Компонент управляет процессом смены сетевого плагина (CNI) в кластере DKP и фиксирует текущее состояние в кастомном ресурсе `CNIMigration`. Поддерживается миграция на плагины `Flannel`, `Simple bridge`, `Cilium`. Подробнее с переключением CNI в кластере можно ознакомиться в соответствующем руководстве.

i Компонент создаётся глобальным хуком `detect-cni-migration` при наличии кастомного ресурса `CNIMigration`. Ресурс `CNIMigration` создаётся администратором вручную или при использовании команды `d8 network cni-migration switch --to-cni <target cni>`.

- 4. **Cni-migration-agent** (DaemonSet) — опциональный компонент, запускающийся на всех узлах кластера и состоящий из одного контейнера **agent**. Компонент следит за кастомным ресурсом `CNIMigration` и управляет кастомным ресурсом `CNINodeMigration`, отражающим текущее состояние миграции на конкретном узле.

i Компонент создаётся глобальным хуком `detect-cni-migration` при наличии кастомного ресурса `CNIMigration`. Ресурс `CNIMigration` создаётся администратором вручную или при использовании команды `d8 network cni-migration switch --to-cni <target cni>`.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - работа с кастомными ресурсами API-группы `deckhouse.io`;
 - отслеживание ресурсов `Pod` и `DaemonSet`, а также перезапуск `Pod` при смене сетевого плагина;
 - отслеживание ресурсов, описанных в кастомном ресурсе `ObjectKeeper`;
 - создание и обновление ресурса `Lease`;
 - создание, удаление, изменение и отслеживание ресурсов, описанных в модулях DKP;
 - авторизация запросов.
2. **Documentation** — обновление документации при добавлении или обновлении модуля DKP.
3. **Хранилище образов** — получение образов компонентов модулей вместе с метаданными, в случае если модуль `registry` установлен в режиме `Unmanaged`.
4. **Модуль registry** — получение образов компонентов модулей вместе с метаданными, в случае если модуль `registry` установлен в одном из следующих режимов: `Direct`, `Proxy` или `Local`.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация и конверсия кастомных ресурсов DKP.
- **Prometheus-main** — сбор метрик с контейнеров `deckhouse` и `webhook-handler`.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

- Документация модуля `deckhouse`
- Документация модуля `documentation`
- Документация модуля `registry`

Механизм очередей

Модуль `deckhouse` реализует ядро Deckhouse Kubernetes Platform (DKP), выполняющее различные операции по управлению платформой с использованием механизма очередей. Подробнее с архитектурой модуля можно ознакомиться в соответствующем разделе документации.

Контроллер Deckhouse реализует очереди `addon-operator` и `marketplace`.

Очереди `addon-operator`

Очереди `addon-operator` — это основной механизм обработки встроенных и внешних модулей DKP. Очередь реализована в `shell-operator` и расширена типами задач `addon-operator`. Контроллер Deckhouse синхронизирует кастомные ресурсы `ModuleConfig` и обновляет глобальные или модульные `values` для `addon-operator`.

Типы задач:

Задача	Назначение
<code>GlobalHookRun</code>	Запуск глобальных хуков (<code>onStartup</code> , <code>beforeAll</code> , <code>afterAll</code> , <code>kubernetes</code> , <code>schedule</code>)
<code>GlobalHookEnableKubernetesBindings</code>	Включение Kubernetes-мониторов из глобальных хуков
<code>GlobalHookWaitKubernetesSynchronization</code>	Блокировка очереди до завершения глобальных хуков с установленным параметром <code>executeHookOnSynchronization: true</code>
<code>GlobalHookEnableScheduleBindings</code>	Регистрация <code>cron</code> -задач в планировщике <code>addon-operator</code>
<code>DiscoverHelmReleases</code>	Поиск «лишних» Helm-релизов после первого <code>converge</code>
<code>ApplyKubeConfigValues</code>	Применение изменений из <code>ModuleConfig</code>
<code>ConvergeModules</code>	Полный цикл <code>converge</code> всех модулей
<code>ModuleRun</code>	Настройка или обновление модуля, включая подзадачи: <code>onStartup</code> → <code>sync</code> → <code>beforeHelm</code> → <code>helm</code> → <code>afterHelm</code>
<code>ParallelModuleRun</code>	Пакетный параллельный запуск модулей
<code>ModuleDelete</code>	Удаление модуля (<code>helm delete</code> , <code>afterDeleteHelm</code>)
<code>ModuleHookRun</code>	Запуск <code>module hook</code> по событию
<code>ModuleEnsureCRDs</code>	Установка CRD модуля
<code>ModulePurge</code>	Удаление неизвестного <code>helm release</code>

Типы очередей `addon-operator`:

Очередь	Имя	Назначение
Main	<code>main</code>	Выполнение глобальных хуков при старте, установка критичных модулей, настройка и удаление модулей
Parallel	<code>parallel_queue_0 ... parallel_queue_19</code>	Параллельный <code>ModuleRun</code> с учётом зависимостей модулей (20 очередей)
Hook queues	Из конфигурации задачи (хука)	Задачи конкретного модуля и глобальные хуки

Каждая очередь — отдельный пайплайн с одним `worker`-узлом, который реализует очередь со следующими свойствами:

- задачи могут вставляться как в конец (`AddLast`), так и в начало (`AddFirst`) очереди;
- выполнение задач происходит с начала очереди;

- поддерживается работа с многофазными операциями, которые используют операции:
 - `AddHeadTasks` — вставить подзадачи перед текущей;
 - `AddTailTasks` — вставить подзадачу в конец очереди после того, как текущая завершится успешно;
 - `AddAfterTasks` — вставить подзадачу сразу после текущей;
- задача выполняется до успеха, если не указано `allowFailure: true` в параметре задачи;
- в случае ошибки выполняется экспоненциальный перезапуск (backoff), начиная с задержки в 5 секунд между попытками.



Если задача не может завершиться успешно и при этом в её параметрах не указано `allowFailure: true`, то такая задача блокирует очередь, в которой она выполняется. Задачи в разных очередях обработки не блокируют друг друга.

При старте контроллер Deckhouse создает очереди `main` и `parallel_queue_0..19` и добавляет в `main` задачи в следующем порядке:

- `GlobalHookRun (onStartup)` — для каждого global hook;
- `GlobalHookEnableScheduleBindings` — для включения планировщика cron;
- `GlobalHookEnableKubernetesBindings` — для включения глобальных задач, отслеживающих ресурсы Kubernetes;
- `GlobalHookWaitKubernetesSynchronization`;
- `ConvergeModules (OperatorStartup)` — первый converge всех модулей.

После `ConvergeModules` контроллер добавляет задачу `DiscoverHelmReleases` — очистка неизвестных helm releases.

Порядок обработки модулей в задаче `ConvergeModules` определяется несколькими атрибутами:

- критичность модуля — заданный параметр `critical` в конфигурации модуля `module.yaml`;
- *вес* модуля — числовое обозначение порядка обработки модуля, чем число выше, тем позже будет обрабатываться модуль. Вес модуля берётся из параметра `weight` конфигурации модуля `module.yaml`, если этого параметра нет или он равен 0, то используется вес по-умолчанию, равный 900. Если файла конфигурации модуля не существует, то используется вес из числового префикса директории с модулем (например: `040-node-manager` — вес 40). Если вес не удалось получить из имени директории, используется вес, равный 100;
- зависимости модуля — список модулей, которые должны быть установлены до текущего модуля.

На основе этих атрибутов планировщик контроллера Deckhouse выстраивает порядок обработки по следующим принципам:

- для критичных модулей учитывается вес модуля в порядке возрастания — задачи помещаются в очередь `main`;
- для некритичных модулей вес модуля не учитывается — задачи помещаются в очереди `parallel_queue_0..19`;
- для всех модулей учитываются зависимости от других модулей.

При этом, если критичные модули могут быть обработаны параллельно, то планировщик контроллера Deckhouse помещает в очередь `main` задачу `ParallelModuleRun` с указанием списка модулей. Задача `ParallelModuleRun` запускает для каждого модуля задачу `ModuleRun` в очередях `parallel_queue_0..19` и ждёт завершения их работы, тем самым блокируя очередь `main`. Если при обработке задачи `ModuleRun` происходит ошибка, то такую задачу планировщик перемещает в конец очереди и запускает следующую задачу из очереди.

Процесс установки критичных модулей изображен на следующей диаграмме:

Процесс установки некритичных (функциональных) модулей изображен на следующей диаграмме:

Если в очередь добавлено более одной идентичной задачи, то при старте выполнения такой задачи все остальные задачи удаляются из очереди для дедупликации.

Для просмотра очередей `addon-operator` можно использовать команду `d8 system queue list`.

Очереди Marketplace

Очереди Marketplace — реализация очереди для работы функционала Marketplace.

Каждая очередь, которую обслуживает контроллер Deckhouse для Marketplace, обладает следующими свойствами:

- FIFO (First In First Out) — определяет неизменяемый порядок выполнения задач. Задача, поступившая в очередь первой, будет исполнена в первую очередь;

- строго последовательное выполнение задач, одна задача за раз;
- запуск задач происходит только при получении событий (event-driven), исключён опрос каких-либо ресурсов;
- повторный запуск при ошибке с экспоненциальным ростом промежутка между попытками, начиная с 15 секунд, но не более 1 минуты, без лимита на количество попыток;
- поддерживается каскадная отмена задач при смене версии или удалении пакета.

Типы очередей:

Имя	Назначение
<code>{packageName}</code>	Реализация задач жизненного цикла пакетов: Deploy, Load, Configure, Enable, Run, Disable, Undeploy
<code>{packageName}/{hookQueue}</code>	Выполнение хуков, запускаемым по Kubernetes-или schedule-событиям (очередь из binding-хука)
<code>{packageName}/{hookQueue}/sync</code>	Синхронизация хука при старте (WaitForSynchronization)

Типы задач:

Задача	Назначение
Deploy	Скачивание или монтирование образа пакета
Load	Парсинг конфигурации, создание Application или Module, регистрация в scheduler
Configure	Применение настроек Application или Module, используя хранилище параметров
Enable	Включение хуков, выполнение синхронизации параметров, запуск хука на событие OnStartup
Run	Выполнение подзадач при установке Application или Module: BeforeHelm → helm Upgrade → AfterHelm
HookRun	Запуск хука по событию
HookSync	Начальная синхронизация Kubernetes binding
Disable	Удаление Helm, отключение хуков, очистка hook-очереди
Undeploy	Удаление пакета с диска

Выполнение задач в одной очереди не блокирует выполнение задач в другой очереди.

Для просмотра очередей Marketplace используйте следующую команду:



Контроллер Deckhouse при выполнении задач по установке или настройке модулей в addon-operator ставит на паузу обработку очередей Marketplace.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `deckhouse`

Модуль registry

Модуль `registry` отвечает за управление настройками хранилища образов компонентов Deckhouse Kubernetes Platform (DKP).

Модуль работает в следующих режимах:

- `Direct` — использование прямого доступа к внешнему хранилищу образов по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse`. Фиксированный адрес позволяет избежать повторного скачивания образов и перезапуска компонентов при смене параметров хранилища образов. Фиксированный адрес заменяется фактическим адресом внешнего хранилища при отправке запросов за счет настройки `mirroring/rewrite` в конфигурации компонента `containerd` и за счет внутреннего `registry-incluster-proxy` компонента, проксирующего запросы к внешнему хранилищу.

- **Proxy** — использование внутреннего кеширующего хранилища образов для проксирования обращений к внешнему хранилищу, с запуском кеширующего прокси хранилища на control plane (master) узлах. Режим позволяет сократить количество запросов к внешнему хранилищу за счёт кеширования образов. Кешируемые данные хранятся на control plane (master) узлах. Обращение к внутреннему хранилищу образов выполняется по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse` аналогично **Direct** режиму.
- **Local** — использование локального внутреннего хранилища образов, с запуском хранилища образов на control plane (master) узлах. Режим позволяет кластеру работать в изолированной среде. Данные хранятся на control plane (master) узлах. Обращение к внутреннему хранилищу образов выполняется по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse` аналогично **Direct** и **Proxy** режимам.
- **Unmanaged** — работа без использования внутреннего хранилища образов. Обращение внутри кластера выполняется напрямую к внешнему хранилищу образов.

Подробнее с описанием настройки и примерами использования модуля можно ознакомиться в разделе документации модуля.

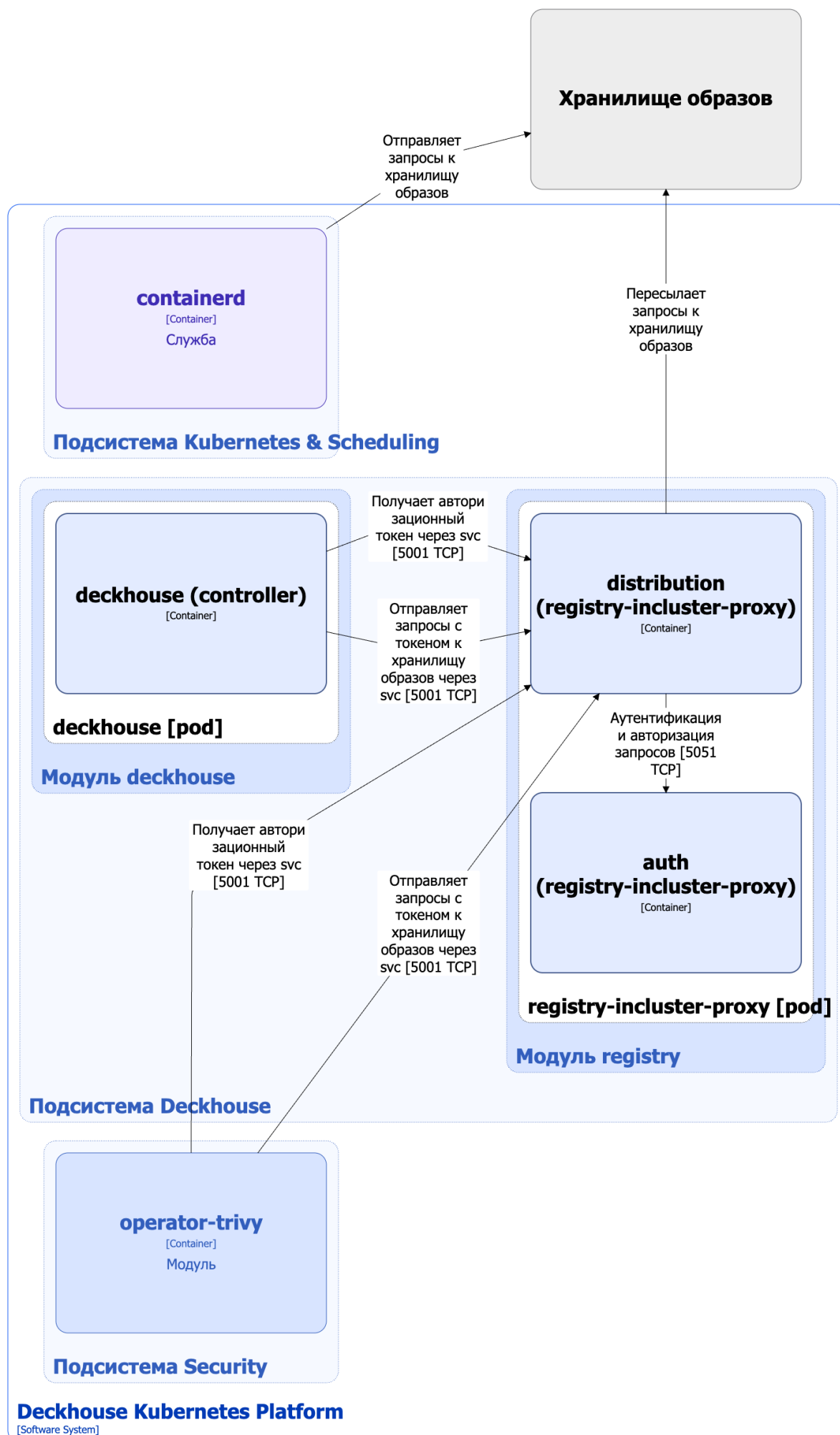
Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `registry` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующих диаграммах:

Модуль `registry` в режиме **Direct**:





- **distribution** — основной контейнер;
 - **auth** — сайдкар-контейнер аутентификации и авторизации. Подробно описан выше;
 - **mirrorer** (режим `Local`) — сайдкар-контейнер, синхронизирующий образы между репликами `distribution`, запущенными на `control plane (master)` узлах.
3. **Registry-proxy** (режимы `Proxy` или `Local`) — балансировщик, установленный на каждый узел кластера и обеспечивающий высокую доступность к хранилищу образов. CRI обращается к хранилищу образов через балансировщик. Настройки для обращения к балансировщику указываются в конфигурации `containerd`.
- Состоит из одного контейнера:
- **registry-proxy** — контейнер, собранный на базе стандартного образа `NGINX`. В контейнере также запускается процесс `registry-proxy-reloader`, который перезагружает процессы `nginx` балансировщика при изменении его конфигурации.
4. **Registry-nodeservices-manager** (режимы `Proxy` или `Local`) — контроллер, управляющий жизненным циклом компонентов `registry-nodeservices`. `Registry-nodeservices-manager` выполняет следующие операции:
- рендерит манифест статического пода `registry-nodeservices` из шаблонов и сохраняет его в каталоге `/etc/kubernetes` на `control plane (master)` узлах. `Kubelet` обнаруживает созданные манифесты и запускает статический под `registry-nodeservices`;
 - сохраняет в каталоге `/etc/kubernetes/registry` на `control plane (master)` узлах файлы конфигурации, необходимые для работы компонентов `registry-nodeservices`;
 - удаляет манифест статического пода из каталога `/etc/kubernetes` и конфигурационные файлы из каталога `/etc/kubernetes/registry`, тем самым удаляя локальное хранилище образов, в случае переключения модуля в режим `Direct` или `Unmanaged`.

Состоит из следующих контейнеров:

- **registry-nodeservices-manager** — основной контейнер;
- Набор сайдкар-контейнеров для предварительного скачивания образов соответствующих компонентов `registry-nodeservices`. Контейнеры стоят на паузе и выполняют только функцию хранения образов:
 - **image-holder-auth**;
 - **image-holder-distribution**;
 - **image-holder-mirrorer**.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - проверяет что текущий узел является `master`’ом;
 - проверяет, что все контейнеры пода `registry-nodeservices` спулены и под находится в статусе `Ready`;
 - применяет изменения конфигурации статического пода из секрета `registry-node-config-
<Node Name>`.
2. **Внешнее хранилище образов** — модуль перенаправляет внутренние запросы к внешнему хранилищу образов.

С модулем взаимодействуют следующие внешние компоненты:

1. **Containerd** — отправляет запросы к хранилищу образов для скачивания образов, используемых для создания контейнеров.
2. **Deckhouse-controller** — отправляет запросы к хранилищу образов для скачивания образов, используемых для установки модулей.
3. **Operator-trivy (а также другие, компоненты, обращающиеся к хранилищу образов напрямую)** — отправляет запросы к хранилищу образов для скачивания образов, например, для проверки их на безопасность (в случае `operator-trivy`).

Дополнительные ресурсы

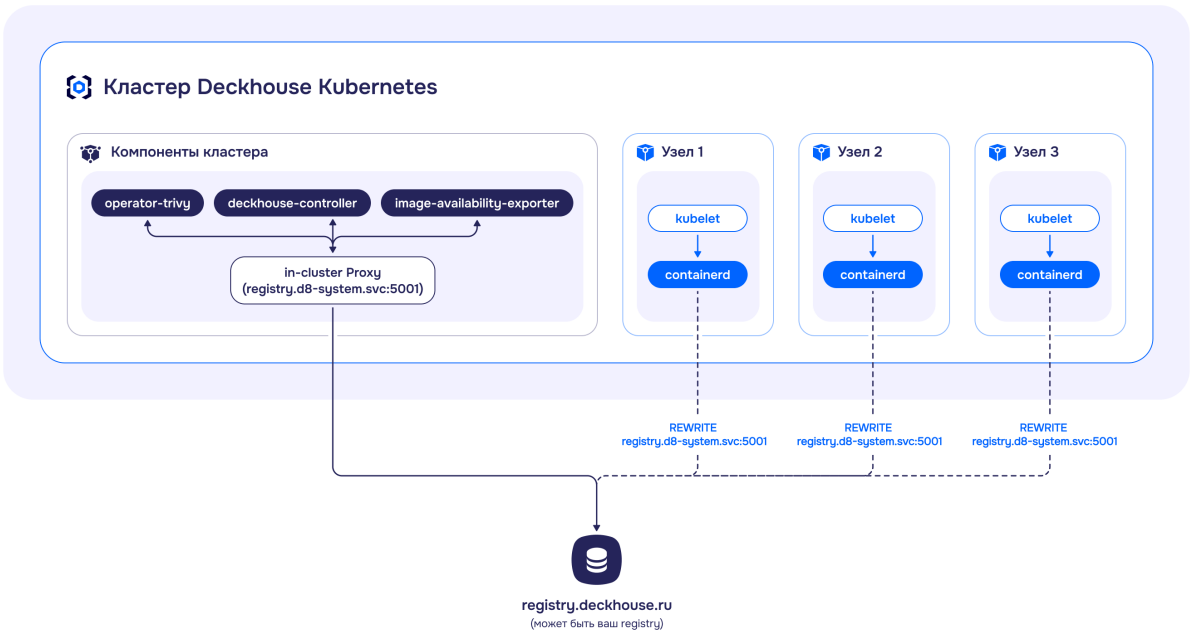
- Параметры модуля `deckhouse`
- Документация модуля `registry`

Архитектура режима Direct

В режиме `Direct` модуля `registry` запросы к хранилищу образов контейнеров обрабатываются напрямую, без промежуточного кеширования.

Перенаправление запросов к хранилищу образов от CRI осуществляется при помощи его настроек, которые указываются в конфигурации `containerd`.

Для таких компонентов, как `operator-trivy`, `image-availability-exporter`, `deckhouse-controller` и ряда других, обращающихся к хранилищу образов напрямую, запросы проходят через внутрикластерный прокси-сервер, расположенный на master-узлах.



Подробнее о режиме `Direct` читайте в разделе об управлении внутренним хранилищем образов.

Дополнительные ресурсы

- Документация модуля `registry`
- Документация модуля `operator-trivy`

Marketplace

Marketplace — это подсистема Deckhouse Kubernetes Platform (DKP), управляющая жизненным циклом единиц поставки, называемых **Packages** (пакетами). Package может быть **Application** (пользовательская нагрузка, развёртываемая в неймспейс) или **Module** (расширение возможностей кластера). В настоящее время поддерживаются только Applications; поддержка Module запланирована на будущую версию.

Marketplace доступен начиная с DKP версии 1.76.

Разделы

В разделе «Концепции» описаны абстракция Package, модель ресурсов, ограничения Application и полный жизненный цикл от сканирования до деплоя.

В разделе «Разработка приложений» описано создание пакета Application с нуля: бутстрап, структура проекта, `package.yaml`, CI/CD и организация артефактов в реестре.

В разделе «Аннотации Nelm» рассматривается полный набор Nelm-аннотаций, используемых в шаблонах Application для управления порядком деплоя, жизненным циклом ресурсов, трекингом и логированием.

В разделе «Хуки» описано написание Go-хуков для Applications с использованием типа `ApplicationHookInput` из `module-sdk`.

Концепции

Типы Package

Package — это абстракция, объединяющая **Application** и **Module**. Различие определяется областью видимости и назначением:

Характеристика	Module	Application
----------------	--------	-------------

Характеристика	Module	Application
Назначение	Инфраструктурное расширение кластера	Пользовательская нагрузка
Область видимости	Cluster-wide (один на кластер)	Namespaced (неограниченное число экземпляров)
Множественные экземпляры	Нет (1:1 с кластером)	Да (N экземпляров в разных неймспейсах)
Включение по умолчанию	Может быть включён через bundle	Только по явному действию пользователя
Создание CRD	Разрешено	Запрещено
Cluster-wide объекты	Разрешены	Запрещены

Модель ресурсов

Marketplace Deckhouse Kubernetes Platform (DKP) использует пять custom resources:

```

flowchart TD
    PR[PackageRepository] -->|инициирует| PRO[PackageRepositoryOperation]
    PR -->|заполняет| APV[ApplicationPackageVersion]
    APV -->|агрегирует| AP[ApplicationPackage]
    APV -->|используется| APP[Application\nnamespace-scoped]

```

Ресурс	Короткое имя	Область	Роль
PackageRepository	—	Cluster	Подключение реестра и расписание сканирования
PackageRepositoryOperation	pro	Cluster	Задача сканирования, обнаруживающая версии
ApplicationPackageVersion	apv	Cluster	Одна на каждую обнаруженную версию пакета; содержит метаданные, OpenAPI-схемы и требования
ApplicationPackage	—	Cluster	Информационный агрегат: какие репозитории содержат пакет, сколько экземпляров используют его
Application	app	Namespace	Установленный экземпляр; управляет деплоем через Helm

Содержимое ApplicationPackageVersion

Каждый объект ApplicationPackageVersion содержит:

- `status.packageMetadata.description` — локализованное описание пакета (en/ru)
- `status.packageMetadata.category` — категория в каталоге
- `status.packageMetadata.stage` — стадия зрелости (Preview, General Availability и т. д.)
- `status.packageMetadata.requirements` — ограничения на версии DKP и Kubernetes; зависимости от модулей (mandatory, conditional, anyOf, noneOf)
- `status.packageMetadata.versionCompatibilityRules` — правила совместимости для обновлений и даунгрейдов
- `status.packageSchemas.settingsSchema` — OpenAPI v3 схема для валидации Application.spec.settings
- `status.packageSchemas.valuesSchema` — OpenAPI v3 схема для effective values, передаваемых в хуки и шаблоны

Жизненный цикл от сканирования до деплоя

1. Администратор создаёт PackageRepository.
2. DKP автоматически создаёт PackageRepositoryOperation (первое сканирование при создании, затем каждые scanInterval).

3. Операция сканирует реестр и создаёт объекты `ApplicationPackageVersion` для каждой обнаруженной версии.
4. Пользователь создаёт `Application` в своём неймспейсе, указывая `packageName`, `packageVersion` и опционально `packageRepositoryName`.
5. DKP проверяет `spec.settings` по `settingsSchema` из соответствующего `ApplicationPackageVersion`.
6. Helm разворачивает Helm-шаблоны из `bundle` пакета.
7. Условия (`conditions`) ресурса `Application` отражают прогресс деплоя: `Installed` → `ConfigurationApplied` → `Scaled` → `Ready`.

Ограничения Application

Все ограничения обеспечивают изоляцию неймспейса и предотвращают влияние `Applications` на ресурсы уровня кластера.

Функциональные ограничения

1. **Запрет создания CRD** — шаблоны `Application` не должны содержать объекты `CustomResourceDefinition`.
2. **Запрет cluster-wide объектов** — все ресурсы, создаваемые `Application`, должны быть `namespaced`.
3. **Нет зависимостей от других Applications** — `Application` может объявлять зависимости только от модулей (через `requirements.modules` в `package.yaml`), но не от других `Applications`.
4. **Хуки ограничены неймспейсом** — хуки не должны читать или записывать ресурсы вне своего неймспейса.
5. **Только явная установка** — `Applications` никогда не активируются по умолчанию; установка требует явного действия пользователя.

Ограничения на имена

Kubernetes ограничивает длину имён Pod до 63 символов. Имя пода `Application` состоит из:

- имени экземпляра — ≤ 24 символа
- имени ресурса — ≤ 24 символа
- суффикса `Deployment` — 15 символов

Поэтому:

- **Имя экземпляра `Application` (`metadata.name`): не более 24 символов**
- **Имя ресурса внутри `Application` (например, суффикс имени `Deployment`): не более 24 символов**

Пример: экземпляр `redis-cache` (11 символов) + ресурс `master-deployment` (17 символов) + суффикс (15 символов) = 43 символа — укладывается в ограничение 63 символа.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Разработка приложений

Предварительные условия

Установите `deckhouse-cli` (d8):

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/deckhouse/deckhouse-cli/main/tools/install.sh)"
```

Войдите в реестр пакетов с помощью лицензионного токена:

```
d8 dk cr login -u license-token dev-registry.deckhouse.io --password <BAШ_TOKEN>
```

Бутстрап нового Application

`d8 package bootstrap application <name>` создаёт директорию `<name>/` в текущей рабочей директории со скелетом пакета и инициализирует git-репозиторий с первым коммитом.

```
d8 package bootstrap application myapp --hooks
cd myapp
git remote add origin <gitlab-repo.git>
git push --set-upstream origin main
```

Флаги:

Флаг	Описание
--hooks	Сгенерировать скелет Go-хуков
--werf	Использовать werf для сборки образов
--extended	Добавить расширенный набор файлов
-o, --output <path>	Кастомный путь вывода (по умолчанию: <cwd>/<name>)

Структура проекта

```
myapp/
├── .gitignore
├── .gitlab-ci.yml          # Пайплайн CI/CD
├── changelog.yaml
├── docs/
│   └── README.md          # Документация приложения
├── hooks/                 # Go-хуки
│   ├── hooks.yaml
│   └── batch/
│       ├── go.mod
│       ├── go.sum
│       ├── main.go
│       └── triggers/
│           └── hook.go
├── images/                # Исходный код образов или инструкции для их загрузки
│   └── myapp/
│       └── werf.inc.yaml
├── openapi/
│   ├── config-values.yaml # OpenAPI-схема для Application.spec.settings
│   └── values.yaml        # OpenAPI-схема для Helm values
├── oss.yaml
├── package.yaml           # Манифест пакета
└── templates/            # Helm-шаблоны
    ├── deployment.yaml
    ├── registry-secret.yaml
    └── service.yaml
```

package.yaml

Центральный манифест пакета Application. Определяет метаданные, тип, требования и совместимость.

```
apiVersion: v1
type: "Application"
name: redis
descriptions:
  ru: "Redis – in-memory база данных"
  en: "Redis – in-memory database"
version: "v1.0.1"      # Инжектируется автоматически при сборке.
stage: "Preview"
category: "Databases"
# Требования к окружению.
requirements:
  deckhouse:
    constraint: ">= 1.70"
  kubernetes:
    constraint: ">= 1.31"
  modules:
    mandatory:
      - name: cert-manager
        constraint: ">= 1.0.0"
```

Справочник полей:

Поле	Обязательное	Описание
name	Да	Уникальное имя пакета

Поле	Обязательное	Описание
descriptions	Да	Локализованное описание для каталога и UI (ru, en)
version	Да	Semver-версия; инжектируется при сборке
type	Да	Application или Module
stage	Да	Стадия зрелости (Preview, General Availability и т. д.)
category	Да	Категория для классификации в каталоге
requirements.deckhouse	Нет	Ограничение на минимальную версию DKP
requirements.kubernetes	Нет	Ограничение на минимальную версию Kubernetes
requirements.modules	Нет	Зависимости от модулей (semver-ограничения)

Локальная сборка

Сборка и публикация пакета в реестр:

```
d8 package build -V v0.0.1 -r dev-registry.deckhouse.io/deckhouse/packages
```

Для локальной разработки используйте модуль payload-registry в качестве личного реестра.

Линтинг

Проверка структуры и конфигурации пакета:

```
d8 package verify
```

Выводит ошибки и предупреждения на основе .pkglint.yaml и встроенных правил.

Настройка CI/CD

Переменные окружения

Переменная	Описание
PACKAGES_REGISTRY_LOGIN	Логин для публикации в реестр
PACKAGES_REGISTRY_PASSWORD	Пароль или токен реестра

Выпуск релиза

Пайплайн запускается по semver git-тегу:

```
git tag v0.1.0
git push origin v0.1.0
```

Пайплайн собирает пакет и публикует его в реестр. После завершения пайплайна версия пакета становится доступной для сканирования через PackageRepository.

Организация OCI-артефактов в реестре


```
registry.deckhouse.io/deckhouse/<edition>/packages:<name>
  Тер с именем пакета — для поддержки листинга

registry.deckhouse.io/deckhouse/<edition>/packages/<name>:<version>
  Bundle — содержит шаблоны, openapi/, hooks/

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/extra/<image>:<version>
  Дополнительные образы (контейнеры приложения)

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/version:<version>
  Метаданные версии — содержат package.yaml, version.json, changelog.yaml

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/version:<release-channel>
  Рекомендуемая версия для release channel
```

Содержимое bundle

Основной образ bundle (`<name>:<version>`) содержит:

```
├─ package.yaml      # Манифест пакета
├─ openapi/          # Схемы settings и values
├─ templates/        # Helm-шаблоны
└─ hooks/            # Хуки жизненного цикла
```

Содержимое образа метаданных

Образ метаданных версии (`<name>/version:<version>`) содержит:

```
├─ package.yaml      # Манифест пакета
├─ version.json       # Semver-версия
└─ changelog.yaml    # Release notes
```

Аннотации Nelm

Шаблоны Application рендерятся и разворачиваются с помощью **Nelm**. Nelm расширяет стандартное поведение Helm аннотациями, управляющими порядком деплоя, жизненным циклом ресурсов, трекингом готовности и выводом логов. На этой странице описаны все аннотации, доступные в шаблонах Application.

Стадии деплоя

Nelm обрабатывает деплой в три стадии. Разные группы аннотаций влияют на разные стадии.

1. Render

Шаблоны вычисляются с текущими values. Доступ к кластеру не нужен (кроме `lookup`). На этой стадии:

- `werf.io/deploy-on` управляет включением ресурса в результат рендера для текущего типа операции (`install`, `upgrade` и т. д.).
- Выполняются шаблонные функции (`werf_secret_file`, `dump debug` и др.).
- Расшифровываются `secret-values.yaml` и файлы из `secret/`.

2. Plan

Nelm подключается к кластеру, читает текущее состояние ресурсов и запускает **dry-run Server-Side Apply** для вычисления точного diff. Затем строит **DAG операций** — какие ресурсы создать, обновить или удалить и в каком порядке.

На этой стадии:

- Аннотации порядка и зависимостей (`werf.io/weight`, `werf.io/deploy-dependency-*`, `*.external-dependency.werf.io/*`) формируют DAG.
- Аннотации жизненного цикла (`werf.io/ownership`, `werf.io/delete-policy`, `werf.io/delete-propagation`) определяют, какие операции попадут в план.

Dry-run SSA означает, что diff вычисляет API server — defaulting, admission-плагины и мутирующие webhook'и применяются корректно.

3. Apply

Выполняется DAG: ресурсы создаются, обновляются или удаляются в порядке зависимостей, с параллелизмом там, где зависимостей нет. Трекинг готовности и стриминг логов выполняются

параллельно.

На этой стадии:

- Аннотации трекинга (`werf.io/track-termination-mode`, `werf.io/fail-mode`, `werf.io/failures-allowed-per-replica`, `werf.io/no-activity-timeout`) управляют ожиданием.
- Аннотации логов управляют выводом во время деплоя.

Типичные задачи

1. Job, которая должна запуститься перед основным приложением

Классический сценарий: миграция БД перед запуском приложения.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: {{.Application.Instance.Name}}-db-migrate
  annotations:
    werf.io/delete-policy: before-creation
spec: ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Application.Instance.Name}}-app
  annotations:
    werf.io/deploy-dependency-migrate: state=ready,kind=Job,name=db-migrate
spec: ...
```

`before-creation` гарантирует пересоздание Job при каждом деплое (иначе Kubernetes откажет в обновлении иммутабельных полей Job). Deployment ждёт `ready` — то есть успешного завершения Job.

2. Сохранение ресурса при удалении или `uninstall`

Сценарий: PVC с данными БД, который не должен удаляться при `uninstall Application`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{.Application.Instance.Name}}-postgres-data
  annotations:
    helm.sh/resource-policy: keep
spec:
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 50Gi
  storageClassName: gp3
```

`helm.sh/resource-policy: keep` предотвращает удаление PVC при `uninstall` или удалении из чарта. От `d8 k delete pvc` не защищает — для этого используйте `persistentVolumeReclaimPolicy: Retain` на StorageClass.

3. Ресурс, общий между релизами

TLS-секрет, используемый несколькими чартами, не должен исчезать при `uninstall` любого из них.

```

apiVersion: v1
kind: Secret
metadata:
  name: {{.Application.Instance.Name}}-shared-tls
  annotations:
    werf.io/ownership: anyone
type: kubernetes.io/tls
data: ...

```

`anyone` сообщает Helm, что он не единственный владелец ресурса, поэтому Helm пропускает удаление при `uninstall`.

4. Зависимость от ресурса, созданного оператором

Деплой только после выпуска сертификата `cert-manager`:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Application.Instance.Name}}-app
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/myapp-tls
spec: ...

```

Helm дождётся состояния `present` и `ready` у `Certificate`, прежде чем создавать `Deployment`.

5. Некритичный компонент, не роняющий весь деплой

`DaemonSet` с метриками, чья недоступность не должна блокировать релиз:

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: {{.Application.Instance.Name}}-metrics-agent
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
spec: ...

```

Helm не будет ждать готовности и не упадёт по таймауту.

6. Ресурс, рендерящийся только при первой установке

Init Job, нужна только при `install`, не при `upgrade`:

```

apiVersion: batch/v1
kind: Job
metadata:
  name: {{.Application.Instance.Name}}-init-data
  annotations:
    werf.io/deploy-on: install
    werf.io/ownership: anyone # КРИТИЧНО
spec: ...

```

Без `werf.io/ownership: anyone` при `upgrade` ресурс рендерится как отсутствующий, и владеющий релиз его удалит. `anyone` предотвращает это.

7. Медленно стартующий ресурс с большим образом

`StatefulSet` с большим Docker-образом (ML-модели, Elasticsearch с предзагруженными индексами):

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: elasticsearch
  annotations:
    werf.io/no-activity-timeout: 20m
    werf.io/failures-allowed-per-replica: "3"
    werf.io/show-service-messages: "true"
spec: ...

```

- `no-activity-timeout: 20m` — 20 минут без событий, прежде чем Nelm считает это таймаутом. Используйте для медленного image pull или долгой инициализации.
- `failures-allowed-per-replica: "3"` — допускает до 3 перезапусков на реплику перед объявлением ошибки. Используйте для нестабильных зависимостей при инициализации.
- `show-service-messages: "true"` — выводить Kubernetes Events в лог деплоя (полезно для диагностики `ImagePullBackOff`, ошибок планирования, OOM).

Порядок и зависимости

Веса

`werf.io/weight` группирует ресурсы: одинаковые веса деплоятся параллельно; разные — последовательно в порядке возрастания.

```

metadata:
  annotations:
    werf.io/weight: "-10" # Деплоится раньше ресурсов с весом по умолчанию (0)

```

Прямые зависимости

`werf.io/deploy-dependency-<id>` ждёт, пока конкретный ресурс в том же релизе достигнет указанного состояния, прежде чем деплоить аннотированный ресурс.

```

metadata:
  annotations:
    werf.io/deploy-dependency-db: state=ready,kind=StatefulSet,name=postgres
    werf.io/deploy-dependency-migrations: state=present,kind=Job,name=db-migrate

```

Состояния зависимости:

- `ready` — ресурс в готовом состоянии (например, у Deployment сошлись `availableReplicas == replicas`).
- `present` — ресурс существует в кластере.

Полный формат:

```

werf.io/deploy-dependency-<id>: state=ready|present[,name=<name>][,namespace=<namespace>][,kind=<kind>][,group=<group>][,version=<version>]

```



Аннотация не работает, если ресурс-зависимость находится в другой стадии деплоя (pre/main/post). Порядок между стадиями уже обеспечивает сама последовательность стадий.

Внешние зависимости

`<id>.external-dependency.werf.io/resource` ждёт ресурс **вне релиза** (созданный оператором или другим релизом):

```

metadata:
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/мyapp-tls
    cert.external-dependency.werf.io/name: myapp-production # Неймспейс внешнего ресурса

```

Полный формат:

```
<id>.external-dependency.werf.io/resource: <kind>[.<version>.<group>]/<name>
```

Зависимости при удалении

`werf.io/delete-dependency-<id>` — аннотированный ресурс будет удалён только после того, как указанный ресурс станет `absent`:

```
metadata:
  annotations:
    werf.io/delete-dependency-app: state=absent,kind=Deployment,name=app
```

Аннотации жизненного цикла

`helm.sh/resource-policy`

`keep` — не удалять ресурс при `uninstall` или удалении из чарта. Ресурс продолжает обновляться при `install/upgrade`, пока рендерится.

`werf.io/ownership`

- `release` (по умолчанию для обычных ресурсов) — ресурс удаляется при `uninstall` и при отсутствии в чарте. Применяются `release` `metadata`-аннотации.
- `anyone` (по умолчанию для хуков и CRD из `crds/`) — ресурс не удаляется при `uninstall`. `Release` `metadata`-аннотации не применяются.

Используйте `anyone` для ресурсов, общих между релизами, или для ресурсов, которые должны переживать свой релиз (init Job с `werf.io/deploy-on: install`).

`werf.io/deploy-on`

Управляет тем, в каких операциях жизненного цикла рендерится ресурс.

```
werf.io/deploy-on: pre-install,upgrade,post-install
```

Допустимые значения: `pre-install`, `install`, `post-install`, `pre-upgrade`, `upgrade`, `post-upgrade`, `pre-rollback`, `rollback`, `post-rollback`, `pre-uninstall`, `uninstall`, `post-uninstall`. По умолчанию: `install,upgrade,rollback`.



Если ресурс рендерится для `install`, но не для `upgrade`, и у него `werf.io/ownership: release` — ресурс будет **удалён при upgrade**, так как он отсутствует в `upgrade`-рендере. Установите `werf.io/ownership: anyone`, чтобы это предотвратить.

`werf.io/delete-policy`

Управляет тем, когда ресурс удаляется относительно операции `apply`.

Значение	Когда
<code>before-creation</code>	Всегда пересоздавать перед <code>apply</code>
<code>before-creation-if-immutable</code>	Пересоздавать только при ошибке «поле иммутабельно» (по умолчанию для Job)
<code>succeeded</code>	Удалить после успешного деплоя
<code>failed</code>	Удалить при провале <code>readiness</code> -чека

Значения можно комбинировать через запятую: `before-creation,succeeded`.

`werf.io/delete-propagation`

Стратегия каскадного удаления в Kubernetes.

Значение	Поведение
<code>Foreground</code> (по умолчанию)	Ждать удаления зависимых объектов

Значение	Поведение
Background	Удалить ресурс сразу; зависимые удаляются асинхронно
Orphan	Удалить ресурс; зависимые оставить

Аннотации трекинга

Аннотация	По умолчанию	Описание
werf.io/track-termination-mode	WaitUntilResourceReady	WaitUntilResourceReady или NonBlocking
werf.io/fail-mode	FailWholeDeployProcessImmediately	FailWholeDeployProcessImmediately или IgnoreAndContinueDeployProcess
werf.io/failures-allowed-per-replica	1	Число перезапусков на реплику до объявления ошибки
werf.io/no-activity-timeout	4m	Go duration; таймаут при отсутствии событий или изменений статуса
werf.io/show-service-messages	false	Показывать Kubernetes Events в выводе деплоя

Аннотации логов

Аннотация	По умолчанию	Описание
werf.io/skip-logs	false	Скрыть все логи подов
werf.io/skip-logs-for-containers	—	Список контейнеров через запятую для скрытия
werf.io/show-logs-only-for-containers	—	Список контейнеров через запятую; показывать только для них
werf.io/show-logs-only-for-number-of-replicas	1	Показывать логи только для первых N реплик
werf.io/log-regex	—	RE2-паттерн; показывать только совпадающие строки
werf.io/log-regex-skip	—	RE2-паттерн; скрывать совпадающие строки
werf.io/log-regex-for-<container>	—	Per-container RE2-фильтр на показ
werf.io/log-regex-skip-for-<container>	—	Per-container RE2-фильтр на скрытие

Шаблонные функции

werf_secret_file

Встроить расшифрованное содержимое файла из директории `secret/`:

```
data:
  config.yaml: {{ werf_secret_file "config.yaml" | b64enc }}
```

Файлы в `secret/` хранятся зашифрованными (AES-128-CBC, ключ из `NELM_SECRET_KEY`). При рендере расшифровываются в памяти. Используйте для сертификатов, приватных ключей и больших конфигов, которые неудобно хранить в `secret-values.yaml`.

dump_debug, printf_debug, include_debug, tpl_debug

Функции отладочного вывода, которые пишут в логи, не влияя на результат рендера. Активируются

только при debug log-level.

```
{{ dump_debug $ }}
{{ printf_debug "replicaCount: %d" .Values.replicaCount }}
{{ include_debug "myapp.labels" . | nindent 4 }}
{{ tpl_debug "{{ .Values.template }}" . }}
```

Известные подводные камни

1. **null-значения и SSA.** Server-Side Apply часто падает на полях со значением `null`. Если `.Values.foo` равно `nil`, в манифесте появится `foo: null`. Решение: guard-условия `{{ if .Values.foo }}` или использование `default`.
2. **Время выполнения lookup.** Если ресурсы кластера изменились между стадиями Plan и Apply, отрендеренный план может быть устаревшим. Избегайте `lookup` для критичной логики — передавайте данные через `values`.
3. **Недетерминированные функции.** `now`, `randAlphaNum` и итерация по `map` с непостоянным порядком ключей дают разный результат при каждом вызове. С замораживанием плана это может создавать ложные diff'ы.
4. **werf.io/deploy-dependency-* и стадии.** Аннотация не работает через стадии деплоя (`pre/main/post`). Порядок между стадиями обеспечивается самой последовательностью стадий.

Хуки

Хуки Application пишутся на Go с использованием того же `module-sdk`, что и хуки модулей Deckhouse Kubernetes Platform (DKP). Ключевое отличие в том, что хуки Application используют `ApplicationHookInput` вместо `HookInput` — это добавляет возможности, специфичные для Application, и обеспечивает изоляцию неймспейса.

ApplicationHookInput

`ApplicationHookInput` предоставляет все стандартные методы `hook input`, а также:

- **Instance()** — возвращает метаданные об экземпляре Application, включая его имя и неймспейс. Используйте для ограничения операций нужным неймспейсом.
- **ObjectPatcher()** — возвращает `ObjectPatcher`, ограниченный неймспейсом. Операции создания и патча ограничены неймспейсом экземпляра Application. Хуки не могут создавать или изменять cluster-wide ресурсы.

Пример: базовый хук

```
package main

import (
    "context"

    "github.com/deckhouse/module-sdk/pkg"
    applicationhook "github.com/deckhouse/module-sdk/pkg/app-hook"
)

func main() {
    applicationhook.Run(onSync)
}

func onSync(ctx context.Context, input applicationhook.ApplicationHookInput) error {
    instance := input.Instance()
    // instance.Name — имя ресурса Application
    // instance.Namespace — неймспейс, в котором установлен Application

    patcher := input.ObjectPatcher()
    // patcher работает только внутри instance.Namespace

    return nil
}
```

Хук валидации настроек

Application может содержать хук валидации настроек, выполняющийся до того, как DKP применяет изменения. Используется когда валидации через OpenAPI-схему недостаточно — например, для проверки бизнес-логики между несколькими полями настроек.

Хук реализует функцию `Check` с сигнатурой:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result
```

`settingscheck.Input` предоставляет доступ к значениям `Application.spec.settings`.

`settingscheck.Result` — одно из:

- `settingscheck.Allow(warnings...)` — настройки валидны; опционально можно добавить предупреждения.
- `settingscheck.Reject(reason)` — настройки невалидны; ресурс Application не будет применён.

Пример:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result {
    replicas := input.Settings.Get("replicas").Int()
    if replicas == 0 {
        return settingscheck.Reject("replicas cannot be 0")
    }

    var warnings []string
    if replicas == 2 {
        warnings = append(warnings, "an even number of replicas may cause split-brain in some configurations")
    }

    if replicas > 3 {
        return settingscheck.Reject("replicas cannot be greater than 3")
    }

    return settingscheck.Allow(warnings...)
}
```

Гарантия изоляции неймспейса

`ObjectPatcher`, возвращаемый `ApplicationHookInput`, обеспечивает, что:

- Все операции `Create`, `Patch` и `Update` направлены в неймспейс самого Application.
- Обращения к другим неймспейсам или cluster-wide ресурсам отклоняются в рантайме.

Это архитектурное ограничение, а не просто соглашение. Хуки, которым нужно читать (но не писать) cluster-wide ресурсы, могут по-прежнему использовать `d8 k` или Kubernetes API-клиент напрямую, но запись за пределами неймспейса заблокирована.

Подсистема Delivery

В данном подразделе описывается архитектура подсистемы Delivery в Deckhouse Kubernetes Platform (DKP).

В подсистему Delivery входят следующие модули:

- `operator-argo` — управляет инсталляциями ArgoCD в DKP;
- `operator-helm` — обеспечивает декларативное управление развёртыванием Helm-чартов.

Модуль operator-argo

Модуль `operator-argo` разворачивает Argo CD Operator в кластере Deckhouse Kubernetes Platform (DKP). Модуль позволяет установить Argo CD в кластере DKP с помощью ресурса ArgoCD.

Модуль работает со следующими кастомными ресурсами:

- `Application` — описание развёртывания приложений и управление ими;
- `ApplicationSet` — шаблонизация и массовое создание приложений по определённым правилам;
- `AppProject` — определение набора приложений и политик доступа к приложениям;
- `ArgoCD` — основной ресурс для развёртывания и настройки экземпляра Argo CD;
- `ArgoCDExport` — экспорт настроек и состояния Argo CD для резервного копирования или миграции;
- `ImageUpdater` — автоматическое обновление образов контейнеров приложений;

- **NamespaceManagement** — определение правил управления неймспейсами для экземпляра Argo CD;
- **NotificationsConfiguration** — определение параметров уведомлений о событиях в Argo CD и приложениях.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.
- На схеме изображены только основные контейнеры каждого компонента.

Архитектура модуля `operator-argo` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующих диаграммах:

- Основной оператор модуля:
- Вариант развёртывания экземпляра Argo CD с базой данных Redis в неотказоустойчивой конфигурации:
- Вариант развёртывания экземпляра Argo CD с отказоустойчивой конфигурацией Redis (на диаграмме отражены только отличия от основного варианта развёртывания):
- Вариант развёртывания экземпляра Argo CD в управляющем кластере в мультикластерной конфигурации (на диаграмме отражены только отличия от основного варианта развёртывания):

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Argocd-operator-controller-manager** (Deployment) — реализация Argo CD Operator, позволяющая разворачивать экземпляры Argo CD в кластере DKP. Компонент работает со следующими кастомными ресурсами:
 - **ArgoCD** — основной ресурс для развёртывания и настройки экземпляра Argo CD;
 - **ArgoCDExport** — экспорт настроек и состояния Argo CD для резервного копирования или миграции. Оператор читает кастомный ресурс **ArgoCDExport** и создаёт Job/CronJob с именем, совпадающим с именем ресурса **ArgoCDExport**. Созданный Job/CronJob выполняет резервное копирование настроек экземпляра Argo CD;
 - **NamespaceManagement** — определение правил управления неймспейсами для экземпляра Argo CD. Оператор следит за кастомным ресурсом **NamespaceManagement** и соответствующим образом обновляет ConfigMap `argocd-cmd-params-cm`;
 - **NotificationsConfiguration** — определение параметров уведомлений о событиях в Argo CD и приложениях. Оператор читает кастомные ресурсы **NotificationsConfiguration** и на основе них обновляет конфигурацию в ConfigMap `argocd-notifications-cm`.

Argocd-operator-controller-manager создаёт ресурсы Deployment, Secret, ConfigMap, StatefulSet и другие для каждого кастомного ресурса ArgoCD, добавляя имя этого ресурса в качестве префикса для создаваемых ресурсов.

Состоит из следующих контейнеров:

- **manager** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам `manager`.

i Следующие компоненты описывают ресурсы, которые создаёт `argocd-operator-controller-manager` на основе конфигурации, заданной в кастомном ресурсе ArgoCD. Для описания используется префикс `<ArgoCD name>`, который будет заменяться контроллером на имя ресурса ArgoCD.

2. **<ArgoCD name>-server** (Deployment) — `argocd-server`, основной компонент взаимодействия с экземпляром Argo CD. `argocd-server` предоставляет REST/gRPC API и пользовательский веб-интерфейс для управления Argo CD. Компонент позволяет управлять кастомными ресурсами `Application`, `ApplicationSet` и `AppProject` через предоставляемые интерфейсы (веб, API, CLI).

Оператор создаёт компонент, если в параметре `.spec.server.enabled` кастомного ресурса ArgoCD задано значение `true` (значение по умолчанию — `true`).

Состоит из следующих контейнеров:

- **argocd-server-init** — опциональный набор `init`-контейнеров, задаваемых пользователем в параметре `.spec.server.initContainers` кастомного ресурса ArgoCD;
- **rollout-extension** — опциональный `init`-контейнер, загружающий расширение UI для работы с кастомным ресурсом `Rollout`. Модуль не предоставляет контроллер, обрабатывающий этот кастомный ресурс. Контроллер должен быть установлен и настроен отдельно. `argocd-operator-controller-manager` добавляет `rollout-extension`, если значение параметра `.spec.server.enableRolloutsUI` принимает значение `true`;
- **argocd-server** — основной контейнер;
- **argocd-server-sidecar** — опциональный набор сайдкар-контейнеров, задаваемых пользователем в параметре `.spec.server.sidecarContainers` кастомного ресурса ArgoCD.

3. **<ArgoCD name>-repo-server** (Deployment) — `argocd-repo-server`, компонент, отвечающий за обработку шаблонов, генерацию манифестов приложений и работу с внешними репозиториями, используемыми в Argo CD. `argocd-repo-server` отвечает за синхронизацию манифестов приложений из указанных репозиториях и передачу их в соответствующие компоненты для последующего развёртывания.

Оператор создаёт компонент, если в параметре `.spec.repo.enabled` кастомного ресурса ArgoCD задано значение `true` (значение по умолчанию — `true`).

Состоит из следующих контейнеров:

- **copyutil** — `init`-контейнер, копирующий исполняемые файлы для использования из основного контейнера;
- **argocd-repo-server-init** — опциональный набор `init`-контейнеров, настраиваемых через параметр `.spec.repo.initContainers` кастомного ресурса ArgoCD для подготовки окружения;
- **argocd-repo-server** — основной контейнер, выполняющий операции по генерации и обработке манифестов, а также работу с удалёнными Git-репозиториями приложений;
- **argocd-repo-server-sidecar** — опциональный набор сайдкар-контейнеров, задаваемых пользователем в параметре `.spec.repo.sidecarContainers` кастомного ресурса ArgoCD и позволяющих расширять функциональность `repo-server`.

4. **<ArgoCD name>-application-controller** (StatefulSet) — `argocd-application-controller`, компонент, отвечающий за синхронизацию и управление состоянием приложений, определённых в Argo CD. `argocd-application-controller` обеспечивает идемпотентное применение манифестов Kubernetes, управление процессом развёртывания, отката, автоматического восстановления, а также отслеживание состояния ресурсов в кластере.

Оператор создаёт компонент, если в параметре `.spec.controller.enabled` кастомного ресурса ArgoCD задано значение `true` (значение по умолчанию — `true`).

Состоит из следующих контейнеров:

- **application-controller-init** — опциональный набор `init`-контейнеров, настраиваемых через параметр `.spec.controller.initContainers` кастомного ресурса ArgoCD для подготовки окружения;
- **argocd-application-controller** — основной контейнер, реализующий логику синхронизации кастомных ресурсов `Application` и создаваемых на их основе ресурсов;
- **application-controller-sidecar** — опциональный набор сайдкар-контейнеров, задаваемых пользователем в параметре `.spec.controller.sidecarContainers` кастомного ресурса ArgoCD, которые позволяют расширить стандартные возможности контроллера.

5. **<ArgoCD name>-applicationset-controller** (Deployment) — `argocd-applicationset-controller`, опциональный компонент, состоящий из одного контейнера **applicationset-controller** и отвечающий за управление кастомным ресурсом `ApplicationSet` в Argo CD. Он позволяет автоматически создавать, обновлять или удалять ресурсы `Application` на основе заданных шаблонов и генераторов (например, генераторов `Git`, `List`, `Matrix` и `Cluster`). Это облегчает массовое управление похожими приложениями, которые должны быть развёрнуты в различных окружениях или кластерах.

Оператор создаёт компонент, если в параметре `.spec.applicationSet.enabled` кастомного ресурса ArgoCD задано значение `true` (значение по умолчанию — `true`).

Более подробную информацию о компоненте можно найти в документации `applicationset-controller`.

6. **<ArgoCD name>-argocd-image-updater-controller** (Deployment) — `argocd-image-updater-controller`, опциональный компонент, состоящий из одного контейнера **argocd-image-updater** и предназначенный для автоматического обновления образов контейнеров в приложениях Argo CD при появлении новых версий в хранилищах образов. Компонент отслеживает изменения тегов образов и при обнаружении новой версии обновляет соответствующие ресурсы Application в Argo CD (например, значения тегов образов в манифестах или Helm values) через запрос на слияние (pull request) в Git-репозиторий либо напрямую, в зависимости от выбранного способа работы.

Argocd-image-updater-controller выполняет следующие функции:

- управляет кастомным ресурсом ImageUpdater, описывающим параметры для автоматического обновления образов контейнеров приложений;
- периодически проверяет указанные в приложениях образы контейнеров в поддерживаемых реестрах (Docker Hub, Quay.io, Harbor и др.);
- поддерживает фильтрацию тегов образов по шаблонам и стратегиям обновления (`semver`, `latest` и др.);
- при обнаружении новой версии образа автоматически выполняет write-back (то есть записывает новое значение тега образа) в Argo CD Application или в Git-репозиторий в зависимости от настроенного метода.

Для корректной работы компоненту требуются права доступа к Git-репозиториям и, при необходимости, к приватным хранилищам образов. Учётные данные для доступа к хранилищам образов можно хранить в секретах Kubernetes.

Для включения компонента необходимо задать в параметре `.spec.imageUpdater.enabled` кастомного ресурса ArgoCD значение `true`.

Более подробную информацию о компоненте можно найти в документации `argocd-image-updater`.

7. **<ArgoCD name>-notifications-controller** (Deployment) — опциональный контроллер, состоящий из одного контейнера **argocd-notifications-controller** и реализующий отправку уведомлений о событиях в Argo CD (например, об успешной синхронизации приложения, ошибках развёртывания, изменениях статуса и др.) во внешние системы уведомлений, включая электронную почту, Slack, Microsoft Teams, Telegram, OpsGenie, вебхуки и другие.

Основной оператор `argocd-operator-controller-manager` модуля формирует настройки для уведомлений на основе кастомных ресурсов `NotificationsConfiguration` и сохраняет их в ресурсах `ConfigMap argocd-notifications-cm` и `Secret argocd-notifications-secret`, которые используются контроллером для формирования и отправки уведомлений.

Для включения компонента необходимо задать в параметре `.spec.notifications.enabled` кастомного ресурса ArgoCD значение `true`.

Более подробную информацию о механизмах работы можно найти в документации по Argo CD Notifications.

8. **<ArgoCD name>-dex-server** (Deployment) — `argocd-dex-server`, опциональный компонент для аутентификации пользователей в Argo CD, выступающий в роли OIDC-провайдера (OpenID Connect) на базе Dex. Компонент реализует возможность входа пользователей через различные внешние провайдеры аутентификации (LDAP, GitHub, GitLab, SAML, Azure AD и др.), а также поддерживает работу со статическими пользователями, определёнными в конфигурации Dex.

Состоит из следующих контейнеров:

- **copyutil** — `init`-контейнер, копирующий исполняемые файлы для использования из основного контейнера;
- **dex** — основной контейнер.

Для включения компонента необходимо задать параметры в разделе `.spec.sso.dex` кастомного ресурса ArgoCD.



Для аутентификации пользователей Argo CD в DKP модуль `operator-argo` поддерживает интеграцию с модулем `user-authn` (встроенная аутентификация DKP). Другие внешние провайдеры через Dex в данной конфигурации не используются.

Подробнее с примерами использования модуля `operator-argo` можно ознакомиться в разделе документации модуля.

9. **<ArgoCD name>-redis** (Deployment) — `argocd-redis`, обязательный компонент, состоящий из

одного контейнера **redis** и отвечающий за хранение данных об очередях задач и состоянии сессий в Argo CD. `Argocd-redis` реализует отдельный экземпляр базы данных Redis.

`Argocd-operator-controller-manager` разворачивает этот компонент, если параметр `.spec.ha.enabled` кастомного ресурса ArgoCD принимает значение `false`.

10. **<ArgoCD name>-redis-ha-server** (StatefulSet) — `argocd-redis-ha-server`, обязательный компонент для развёртывания Redis в режиме высокой доступности (HA) в составе Argo CD. Реализует отказоустойчивый кластер Redis с репликацией и автоматическим переключением (failover) с помощью механизма Redis Sentinel.

Состоит из следующих контейнеров:

- **config-init** — `init`-контейнер, подготавливающий конфигурацию для Redis и Sentinel перед запуском основных контейнеров;
- **redis** — основной контейнер, реализующий экземпляр Redis-сервера;
- **sentinel** — вспомогательный контейнер, запускающий Redis Sentinel для мониторинга состояния экземпляров Redis и автоматического переключения на реплику при отказе основного экземпляра.

`Argocd-operator-controller-manager` разворачивает этот компонент, если параметр `.spec.ha.enabled` кастомного ресурса ArgoCD принимает значение `true`.

11. **<ArgoCD name>-redis-ha-haproxy** (Deployment) — `argocd-redis-ha-haproxy`, дополнительный компонент для балансировки нагрузки и распределения трафика к экземплярам кластера Redis (`redis-ha-server`) на основе балансировщика нагрузки/обратного прокси-сервера HAProxy.

Состоит из следующих контейнеров:

- **config-init** — `init`-контейнер, подготавливающий конфигурацию для HAProxy перед запуском основного контейнера;
- **haproxy** — контейнер, работающий в роли прокси-сервера и обеспечивающий прозрачную маршрутизацию запросов клиентов к доступным master/replica-экземплярам Redis, а также автоматизацию переключения между ними при failover.

`Argocd-operator-controller-manager` разворачивает этот компонент, если параметр `.spec.ha.enabled` кастомного ресурса ArgoCD принимает значение `true`.

i

Для реализации безопасного и масштабируемого мультикластера в Argo CD используются следующие компоненты:

- **Principal** — это центральная точка управления: хранит состояние и раздает задания;
- **Agent** — это исполнитель в каждом целевом кластере: применяет манифесты и возвращает результат обработки в Principal.

Такой подход позволяет управлять множеством кластеров без прямого доступа с центрального Argo CD в Kubernetes API каждого кластера, что уменьшает требования к количеству открытых входящих доступов, обеспечивает изоляцию и отказоустойчивость.

12. **<ArgoCD name>-agent-agent** (Deployment) — `argocd-agent-agent`, опциональный компонент, состоящий из одного контейнера **<ArgoCD name>-agent-agent** и отвечающий за выполнение операций над управляемыми ресурсами Kubernetes-кластера по заданию из Argo CD. Компонент устанавливает подключение к Argo CD Principal, синхронизирует приложения и управляет их состоянием на основе команд, поступающих от Argo CD Principal.

Подробнее с архитектурой мультикластерной конфигурации Argo CD можно ознакомиться в документации Argo CD.

`Argocd-operator-controller-manager` разворачивает этот компонент, если параметр `.spec.argoCDAgent.agent.enabled` кастомного ресурса ArgoCD принимает значение `true`. В одном ресурсе ArgoCD не допускается одновременное использование Argo CD Agent и Argo CD Principal.

13. **<ArgoCD name>-agent-principal** (Deployment) — `argocd-agent-principal`, опциональный компонент, состоящий из одного контейнера **<ArgoCD name>-agent-principal** и обеспечивающий работу Argo CD в мультикластерной конфигурации.

При включении этого компонента `argocd-operator-controller-manager` перенастраивает все компоненты, использующие подключение к базе Redis, на использование Redis-прокси. Компонент `argocd-agent-principal` реализует Redis-прокси и маршрутизирует запросы к базе данных на основе анализа ключей Redis: в зависимости от значения ключей запрос направляется или в локальный экземпляр Redis, или в один из удалённых Argo CD Agent.

`Argocd-operator-controller-manager` разворачивает этот компонент, если параметр `.spec.argoCDAgent.principal.enabled` кастомного ресурса ArgoCD принимает значение `true`. В одном ресурсе ArgoCD не допускается одновременное использование Argo CD Agent и Argo CD Principal.

14. **<Export name>** (Job/CronJob) — `argocd-export`, опциональный компонент, реализованный в виде Job или CronJob и создающий под из одного контейнера **argocd-export**. Компонент создаёт резервную копию настроек и состояния экземпляра Argo CD.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Внешние хранилища образов** — получение списка образов.
2. **Внешние репозитории кода:**
 - получение манифестов развёртывания приложения из репозитория;
 - обновление параметров образа контейнера (поле `image`) в Helm-чарте.
3. **Внешний Argo CD Principal:**
 - подключение к управляющему кластеру Argo CD;
 - получение запросов на обработку;
 - передача результатов выполнения запросов.
4. **Kube-apiserver:**
 - управление кастомными ресурсами Application, ApplicationSet, AppProject, ArgoCD, ArgoCDExport, ImageUpdater, NamespaceManagement, NotificationsConfiguration, а также Secret, ConfigMap;
 - управление ресурсами, которые создаются при развёртывании пользовательского приложения, описанного в кастомном ресурсе Application;
 - авторизация запросов на получение метрик.
5. **Модуль user-authn** — перенаправление пользователя для аутентификации.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик, предоставляемых оператором и экземплярами Argo CD.
2. **Внешний Argo CD Agent:**
 - подключение к управляющему кластеру Argo CD;
 - получение запросов на обработку;
 - передача результатов выполнения запросов.

Дополнительные ресурсы

- Документация модуля `operator-argo`
- Документация модуля `user-authn`

Подсистема Kubernetes & Scheduling

В данном подразделе описывается архитектура модулей, входящих в подсистему Kubernetes & Scheduling платформы Deckhouse Kubernetes Platform (DKP).

В подсистему Kubernetes & Scheduling входят следующие модули:

- `control-plane-manager` — основной модуль подсистемы, с помощью которого осуществляется управление компонентами control plane кластера;
- `descheduler` — анализирует состояние кластера и выполняет вытеснение подов в соответствии с активными стратегиями;
- `pod-reloader` — позволяет автоматически перезапускать нагрузку в случае изменения определенных ConfigMap или Secret;
- `vertical-pod-autoscaler` — автоматически корректирует запросы и лимиты ресурсов контейнеров в подах на основе фактического потребления. Архитектура модуля описана на соответствующей странице;
- `adaptive-resource-management` — автоматически подбирает оптимальные resource requests и limits для контейнеров на основе рекомендаций VPA (Vertical Pod Autoscaler).

В подразделе также описывается архитектура control plane и агента kubelet.

Control plane кластера

В Deckhouse Kubernetes Platform (DKP) используется стандартный («vanilla») кластер Kubernetes. Control plane кластера включает в себя следующие базовые компоненты:

1. **kube-apiserver** — API-сервер Kubernetes. Обрабатывает REST-запросы, предоставляет интерфейс доступа к общему состоянию кластера, через который взаимодействуют все остальные компоненты, валидирует ресурсы Kubernetes API и сохраняет их в хранилище **etcd**. Включает следующие контейнеры:
 - **kube-apiserver** — основной контейнер;
 - **kube-apiserver-healthcheck** - сайдкар-контейнер, который позволяет проверять работоспособность **kube-apiserver** без включения анонимной аутентификации и без открытия порта, не прошедшего проверку подлинности. Использует клиентский сертификат для аутентификации на API-сервере. Является Open Source-продуктом.
2. **etcd** — распределённое хранилище типа «ключ-значение», где хранится вся конфигурация и

ресурсы Kubernetes-кластера.

3. **kube-scheduler** — планировщик Kubernetes. Анализирует ресурсы узлов и размещает поды с учетом ограничений и правил, таких как affinity и taints.
4. **kube-controller-manager** — диспетчер контроллеров Kubernetes. Запускает циклы контроллеров, которые отслеживают и корректируют состояние стандартных ресурсов Kubernetes, приводя их к желаемому состоянию. Примеры контроллеров, которые поставляются с Kubernetes: replication controller, endpoints controller, namespace controller и ServiceAccount controller.

Взаимодействие компонентов control plane Kubernetes изображено на схеме архитектуры модуля control-plane-manager.

Kubelet

Kubelet не является компонентом control plane, но играет ключевую роль в работе Kubernetes-кластера.

Kubelet — это агент, который работает на каждом узле Kubernetes-кластера. Он обеспечивает запуск контейнеров в подах и их работу в соответствии со спецификациями. Kubelet непрерывно взаимодействует с kube-apiserver, проверяя и поддерживая состояние узлов и контейнеров. Kubelet также отвечает за запуск компонентов control plane.

Манифесты статических подов

Kubelet запускает компоненты control plane из манифестов статических подов, расположенных в директории /etc/kubernetes/manifests. В Deckhouse Kubernetes Platform kubelet обрабатывает в этой директории только файлы с расширением .yaml или .yml.

Файлы с другими расширениями, например kube-apiserver.backup, kube-apiserver.yaml.bak, swar-файлы редакторов или другие временные файлы, игнорируются. Это предотвращает случайную обработку резервных копий и файлов, не являющихся манифестами, как манифестов статических подов.

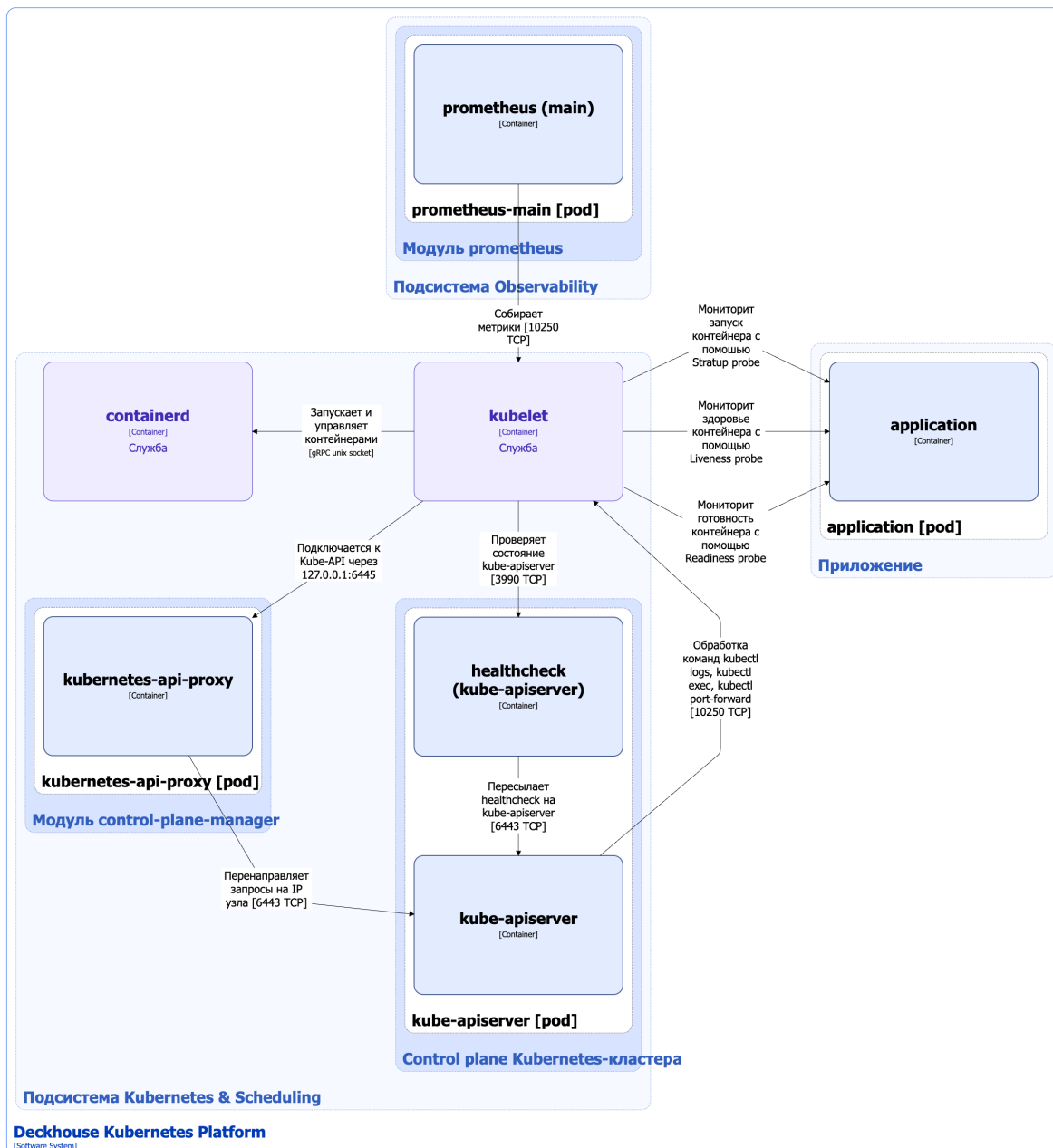
Взаимодействия kubelet

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.
- На схеме показан под application, который представляет собой любой под в кластере: системный и пользовательский.

Взаимодействия kubelet изображены на следующей диаграмме:



Kubelet контролирует состояние контейнеров всех подов, запущенных на узле, относящихся как к пользовательским приложениям, так и к компонентам DKP, выполняя пробы Startup, Liveness и Readiness в соответствии со спецификацией пода. Подробнее о пробах можно узнать в документации Kubernetes.

Kubelet также взаимодействует со следующими компонентами:

1. **Containerd** — отправляет команды для управления жизненным циклом контейнеров на узле, используя для этого Container Runtime Interface (CRI);
2. **Kubernetes-api-proxy** — проксирует запросы к kube-apiserver, отправляемые на адрес localhost. Входит в состав модуля control-plane-manager;
3. **Kube-apiserver-healthcheck** — проверяет состояние kube-apiserver.

С kubelet взаимодействуют следующие компоненты:

1. **Kube-apiserver:**
 - получение логов с подов (обработка команды `kubectl logs`);
 - подключение к запущенным подам (обработка команды `kubectl exec`);
 - переадресация портов (обработка команды `kubectl port-forward`).
2. **Prometheus-main** — собирает метрики kubelet.

Дополнительные ресурсы

- Документация модуля control-plane-manager

Управление компонентами control plane кластера

Модуль control-plane-manager

Управление компонентами control plane кластера осуществляется с помощью модуля `control-plane-manager`, который запускается на всех master-узлах кластера (узлах с лейблом `node-role.kubernetes.io/control-plane: ""`).

Функции управления control plane:

- **Управление сертификатами** — выпуск, продление и обновление сертификатов, необходимых для работы control plane. Позволяет автоматически поддерживать безопасную конфигурацию control plane и оперативно добавлять дополнительные альтернативные имена субъекта (Subject Alternative Name, SAN) для организации защищенного доступа к API Kubernetes.
- **Настройка компонентов** — автоматическое создание необходимых конфигураций и манифестов компонентов control plane.
- **Обновление и откат версий компонентов (upgrade/downgrade)** — поддержание в кластере согласованных версий компонентов.
- **Управление конфигурацией etcd-кластера и его членов** — масштабирование master-узлов и миграция между одномастерными и мультимастерными конфигурациями.
- **Настройка kubeconfig** — поддержание актуальной конфигурации для работы `kubectl` на узлах кластера. Генерация, продление и обновление kubeconfig с правами `cluster-admin`, а также создание символической ссылки для пользователя `root`, чтобы kubeconfig использовался по умолчанию.
- **Расширение работы планировщика** — подключение внешних плагинов через вебхуки с использованием ресурса `KubeSchedulerWebhookConfiguration`. Позволяет использовать продвинутую логику при решении задач планирования нагрузки в кластере, например:
 - размещение подов приложений организации хранилища данных ближе к самим данным;
 - приоритизация узлов в зависимости от их состояния (сетевой нагрузки, состояния подсистемы хранения и т. д.);
 - разделение узлов на зоны, и т. п.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в документации модуля `control-plane-manager`.

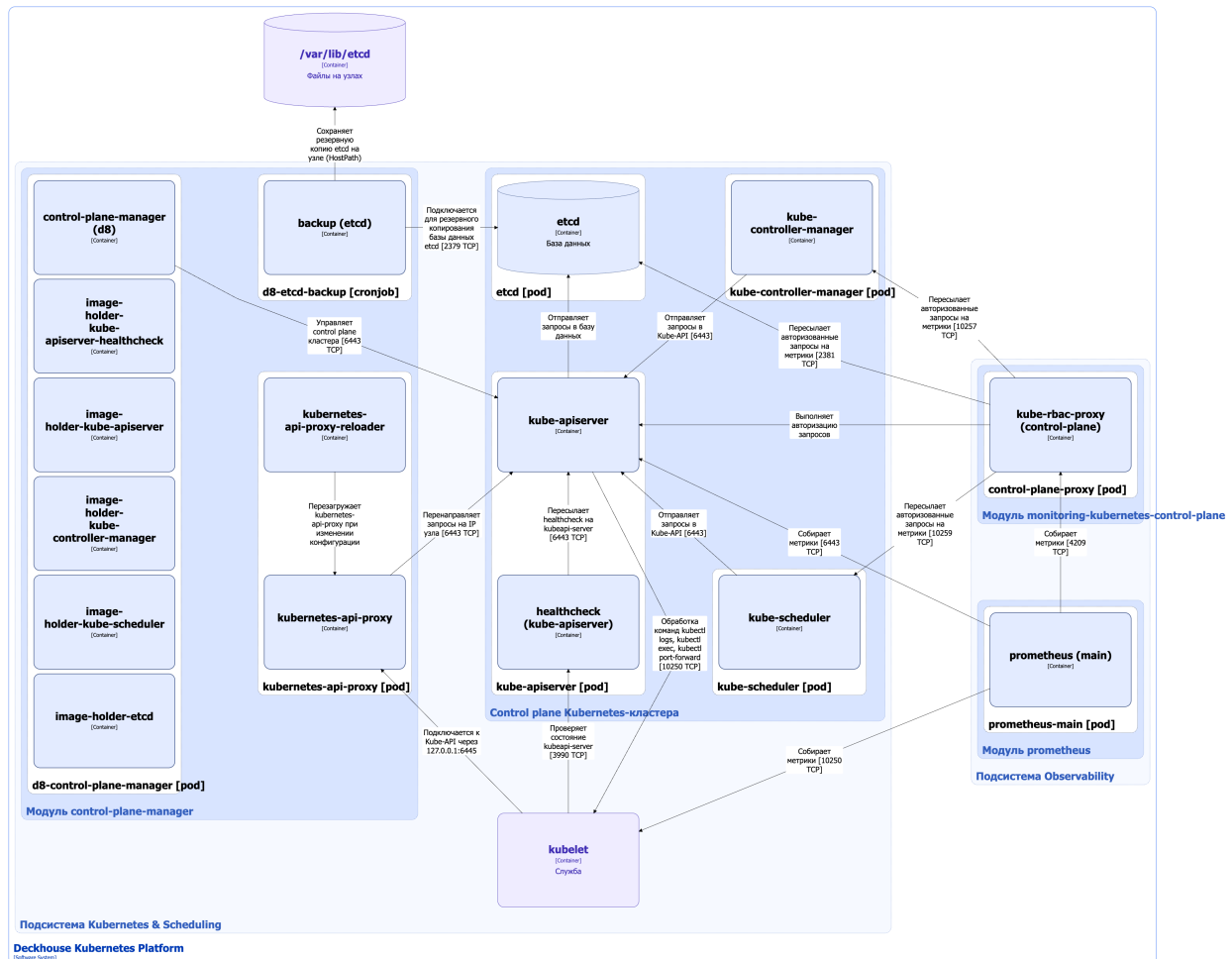
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах. однако на схеме все поды изображены в одной реплике.

Архитектура модуля `control-plane-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **d8-control-plane-manager** (DaemonSet) — управляет компонентами control plane кластера и запускается на всех master-узлах. Состоит из следующих контейнеров:
 - **control-plane-manager** — основной контейнер. Является разработкой компании «Флант».
 - Набор сайдкар-контейнеров для предварительного скачивания образов соответствующих компонентов control plane. Контейнеры стоят на паузе и выполняют только функцию хранения образов:
 - **image-holder-kube-apiserver**;
 - **image-holder-kube-apiserver-healthcheck**;
 - **image-holder-kube-controller-manager**;
 - **image-holder-kube-scheduler**;
 - **image-holder-etcd**.
2. **kubernetes-api-proxy** (статические поды) — на каждом master-узле настраивается дополнительный прокси-сервер, отвечающий на запросы к localhost. По умолчанию проксирует запросы к локальному экземпляру **kube-apiserver**, а в случае его недоступности последовательно опрашивает остальные экземпляры **kube-apiserver**. Включает в себя следующие контейнеры:
 - **kubernetes-api-proxy** — прокси-сервер на базе NGINX;
 - **kubernetes-api-proxy-reloader** — сайдкар-контейнер, перезапускающий прокси-сервер при изменении конфигурации. Является разработкой компании «Флант».
3. **d8-etcd-backup** (CronJob) — периодически выполняет резервное копирование базы данных **etcd** кластера. Состоит из контейнера:
 - **backup** — контейнер с shell-скриптом, который через утилиту `etcdctl` создает снимок базы данных и сохраняет его в каталог `/var/lib/etcd` на master-узле (каталог по умолчанию, может быть изменен через параметры модуля).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **kube-apiserver**:

- управление компонентами control-plane кластера;
- проксирование и балансировка запросов к **kube-apiserver**, отправляемых на адрес `localhost`.

2. **etcd**:

- управление конфигурацией etcd-кластера и его членов;
- периодическое резервное копирование базы данных.

С модулем взаимодействуют следующие внешние компоненты:

1. **kubelet** — запросы к **kube-apiserver**, отправляемые на адрес `localhost`, проксируются компонентом **kubernetes-api-proxy** модуля.

Мониторинг control plane кластера

Мониторинг control plane кластера осуществляется с помощью модуля `monitoring-kubernetes-control-plane`, который обеспечивает безопасный сбор метрик и предоставляет базовый набор правил мониторинга следующих компонентов кластера:

- **kube-apiserver**;
- **kube-controller-manager**;
- **kube-scheduler**;
- **etcd**.

Подробнее с настройками `monitoring-kubernetes-control-plane` можно ознакомиться в документации модуля.

Компоненты модуля `monitoring-kubernetes-control-plane`

Модуль состоит из одного компонента:

1. **control-plane-proxy** (DaemonSet) — запускается на всех master-узлах кластера и состоит из одного контейнера:
 - **kube-rbac-proxy** — авторизующий прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам.

Взаимодействия компонента `control-plane-proxy`

Control-plane-proxy взаимодействует со следующими компонентами:

1. **kube-apiserver** — авторизация запросов на получение метрик.
2. Компоненты control plane кластера — **control-plane-proxy** пересылает авторизованные запросы на метрики до:
 - **kube-controller-manager**;
 - **kube-scheduler**;
 - **etcd**.

С **control-plane-proxy** взаимодействует **prometheus-main** для сбора метрик компонентов control plane.

Взаимодействие модуля `monitoring-kubernetes-control-plane` с control plane кластера изображено на приведенной выше схеме архитектуры модуля `control-plane-manager`.

Сбор метрик с kube-apiserver

Метрики **kube-apiserver** собираются **prometheus-main** напрямую. Модуль `monitoring-kubernetes-control-plane` добавляет правила сбора этих метрик в конфигурацию **prometheus-main**.

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Документация модуля `monitoring-kubernetes-control-plane`

Вертикальное масштабирование

Модуль `vertical-pod-autoscaler` обеспечивает работу Vertical Pod Autoscaler (VPA) в Deckhouse Kubernetes Platform (DKP).

Подробнее о настройках модуля и примерах его использования можно узнать в соответствующем разделе документации.

Режимы работы VPA

VPA может работать в двух режимах:

- Автоматическое изменение запросов ресурсов:
 - **InPlaceOrRecreate** (по умолчанию в Kubernetes, начиная с версии 1.33) — VPA пытается изменить ресурсы без пересоздания подов. Если обновить ресурсы «на месте» (in-place) невозможно, VPA переходит к схеме, аналогичной режиму **Recreate**: под, для которого невозможно обновить ресурсы, вытесняется, и вместо него контроллер создает новый под с обновленными ресурсами.
- i** Чтобы использовать режим **InPlaceOrRecreate** в Kubernetes до версии 1.33, включите экспериментальную функцию (feature gate) `InPlacePodVerticalScaling` в настройках модуля `control-plane-manager`.
- **Auto** (по умолчанию в Kubernetes до версии 1.33) — VPA изменяет ресурсы без пересоздания подов, но при необходимости действует аналогично режиму **Recreate** и перезапускает под. Это устаревший режим, и его поддержка будет прекращена в будущих версиях Deckhouse Kubernetes Platform (DKP).
 - **Recreate** — VPA может изменять ресурсы у работающих подов, перезапуская их. В случае одного пода (`replicas: 1`) это приведет к недоступности сервиса на время перезапуска. VPA не пересоздает поды, если они были созданы без контроллера.
- Только рекомендации, без изменения ресурсов:
 - **Initial** — ресурсы подов изменяются только при их создании, но не в процессе работы.
 - **Off** — VPA не меняет ресурсы автоматически. Однако, с его помощью можно просматривать рекомендуемые ресурсы с помощью команды `d8 k describe vpa`.

При использовании VPA и включении соответствующего режима, запрашиваемые ресурсы устанавливаются автоматически на основе данных из Prometheus. Также возможно настроить систему таким образом, чтобы она только рекомендовала ресурсы, но не изменяла их. Подробнее про включение и настройку VPA можно почитать в разделе «Администрирование».

Ограничения VPA

Перед использованием вертикального масштабирования (VPA) необходимо учитывать ряд ограничений:

- Перезапуск подов при изменении ресурсов:
 - Обновление запрашиваемых ресурсов — экспериментальная функция, каждый раз при изменении ресурсов VPA пересоздаёт под, и он может быть назначен на другой узел;
 - Поды могут пересоздаваться на других узлах.
- Совместимость с Horizontal Pod Autoscaler (HPA):
 - VPA не рекомендуется использовать совместно с HPA, выполняющим масштабирование по CPU или памяти;
 - VPA можно использовать совместно с HPA, выполняющим масштабирование по custom- или external-метрикам.
- Проблемы с большими кластерами — VPA может работать и в больших кластерах, но нагрузка на VPA возрастает при росте числа подов.
- Проблемы с Pending-подами — VPA может рекомендовать ресурсы выше доступных в кластере, из-за чего поды могут застрять в статусе `Pending`.
- Проблемы при удалении VPA — если удалить VPA или отключить его (режим `Off`), ресурсы останутся в последнем измененном значении. Это может привести к путанице, когда в Helm указаны одни ресурсы, в контроллере — другие, а у подов — третьи.
- Использование нескольких VPA-ресурсов на один под — может привести к непредсказуемому поведению.



При использовании VPA рекомендуется настроить Pod Disruption Budget.

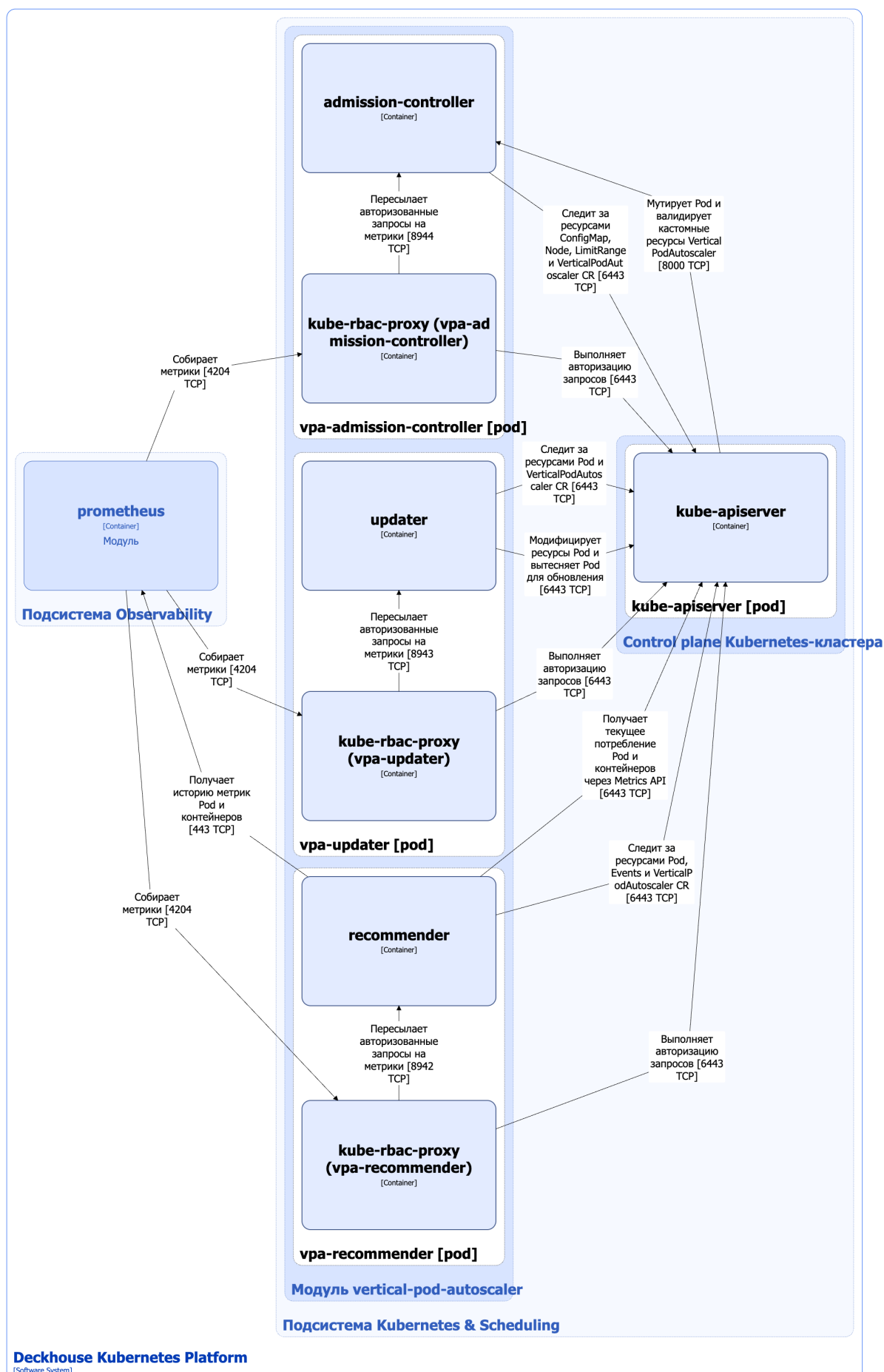
Архитектура модуля



Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `vertical-pod-autoscaler` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP показаны на следующей диаграмме:



Компоненты модуля

Модуль `vertical-pod-autoscaler` состоит из следующих компонентов:

1. **Vpa-admission-controller** (Deployment) — контроллер VPA, обслуживающий работу с

кастомным ресурсом VerticalPodAutoscaler.

Компонент vpa-admission-controller выполняет следующие действия:

- валидирует кастомные ресурсы VerticalPodAutoscaler;
- при создании пода (если для VPA не установлен режим **Off**) контроллер автоматически задаёт или меняет значения `requests` и `limits` в контейнерах, оптимизируя их по текущим рекомендациям. Значения `limits` контроллер изменяет только в том случае, если в параметре `spec.resourcePolicy.containerPolicies.controlledValues` политики управления ресурсами установлено значение `RequestsAndLimits`.

Состоит из следующих контейнеров:

- **admission-controller** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищенного доступа к метрикам admission-controller. Является Open Source-проектом.

2. **Vpa-updater** (Deployment) — компонент VPA, проверяющий, что у подов с VPA выставлены корректные ресурсы. Vpa-updater выполняет in-place-обновление ресурсов через субресурс Kubernetes `pods/resize`, а если это невозможно или не подходит по политике управления ресурсами, вытесняет под.

Состоит из следующих контейнеров:

- **updater** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищенного доступа к метрикам updater. Является Open Source-проектом.

3. **Vpa-recommender** (Deployment) — компонент VPA, определяющий рекомендации для `requests` на основе информации о прошлом и текущем потреблении ресурсов подами.

Vpa-admission-controller и vpa-updater пересчитывают значения `limits` пропорционально `requests` в том случае, если в параметре `spec.resourcePolicy.containerPolicies.controlledValues` политики управления ресурсами установлено значение `RequestsAndLimits`.

Состоит из следующих контейнеров:

- **recommender** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищенного доступа к метрикам recommender. Является Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- наблюдение за стандартными ресурсами ConfigMap, Node, LimitRange, Pod, а также за кастомными ресурсами VerticalPodAutoscaler и VerticalPodAutoscalerCheckpoint;
- получение текущего потребления ресурсов через Metrics API;
- вытеснение работающих подов при несоответствии спецификации ресурсов и рекомендуемых значений;
- авторизация запросов на получение метрик.

2. Prometheus — получение истории метрик потребления ресурсов подом.

С модулем взаимодействуют следующие внешние компоненты:

1. Kube-apiserver:

- валидация кастомных ресурсов VerticalPodAutoscaler;
- изменение `requests` и `limits` в спецификации подов.

2. Prometheus — собирает метрики модуля.

Дополнительные ресурсы

- Включение вертикального масштабирования
- Параметры модуля `control-plane-manager`
- Документация модуля `vertical-pod-autoscaler`

Модуль adaptive-resource-management

Модуль `adaptive-resource-management` позволяет автоматизировать подбор `resource requests` и `limits` для рабочих нагрузок на основе рекомендаций Vertical Pod Autoscaler (VPA).

Для этого модуль разворачивает контроллер **AutoVPA**, основанный на проекте Goldilocks, с доработками Deckhouse Kubernetes Platform (DKP). Контроллер автоматически создает и поддерживает объекты VPA для рабочих нагрузок в выбранных неймспейсах и предоставляет

рекомендации по настройке ресурсов.

Основные возможности:

- Автоматическое создание объектов VPA для Deployment, StatefulSet, DaemonSet и Job в управляемых неймспейсах.
- Гибкий выбор неймспейсов: управление всеми неймспейсами, неймспейсами с определённым лейблом или соответствующими определённому label-селектору.
- Комбинированный режим выбора неймспейсов по label-селектору и специальному лейблу `autovpa.deckhouse.io/enabled`.
- Автоматическое создание объектов VPA в режиме рекомендаций без изменения манифестов приложений.
- Настройка VPA на уровне отдельных нагрузок и неймспейсов через аннотации и лейблы `autovpa.deckhouse.io/*`.
- Минимальное потребление ресурсов: контроллер работает в одной реплике и предъявляет низкие требования к CPU и памяти.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в разделе документации модуля.

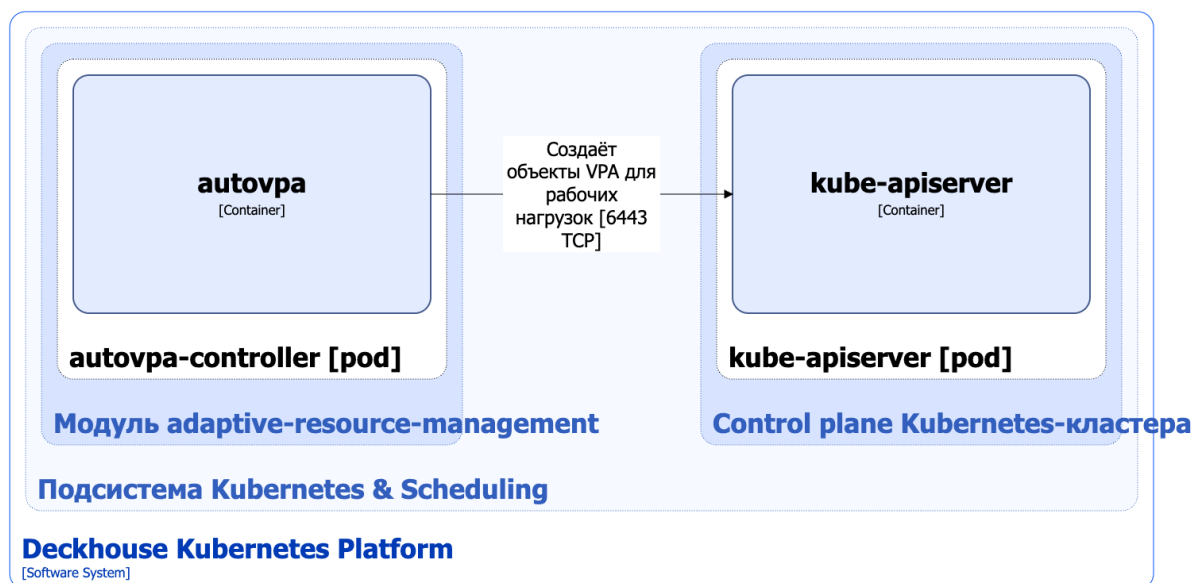
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `adaptive-resource-management` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP показаны на следующей диаграмме:



Компоненты модуля

Модуль `adaptive-resource-management` состоит из одного компонента **autovpa-controller**, включающего основной контейнер **autovpa**.

Взаимодействия модуля

Модуль взаимодействует с компонентом **kube-apiserver** для создания объектов VPA под рабочие нагрузки.

Дополнительные ресурсы

- Документация модуля `adaptive-resource-management`

Модуль descheduler

Модуль `descheduler` обеспечивает работу Descheduler в Deckhouse Kubernetes Platform (DKP).

Модуль периодически анализирует состояние кластера и вытесняет поды, соответствующие условиям активных стратегий. Вытесненные поды вновь проходят процесс планирования с учётом текущего состояния кластера. Это позволяет перераспределить рабочие нагрузки в соответствии с выбранной стратегией.

Если в кластере зарегистрирован сервис Metrics API, модуль использует его для получения информации о текущем потреблении ресурсов.

Параметры стратегий задаются при помощи кастомного ресурса Descheduler. Хук модуля обрабатывает эти ресурсы и пересоздаёт конфигурацию для Descheduler.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `descheduler` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP показаны на следующей диаграмме:

Компоненты модуля

Модуль `descheduler` состоит из одного компонента **descheduler**, включающего следующие контейнеры:

- **descheduler** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищенного доступа к метрикам descheduler.

Взаимодействия модуля

Модуль взаимодействует с компонентом **kube-apiserver**:

- наблюдение за стандартными ресурсами Node и Pod;
- получение текущего потребления ресурсов через Metrics API;
- вытеснение работающих подов для соответствия активным стратегиям;
- авторизация запросов на получение метрик.

С модулем взаимодействует компонент **prometheus** — собирает метрики модуля.

Дополнительные ресурсы

- Документация модуля `descheduler`

Модуль pod-reloader

Модуль `pod-reloader` создан на основе Reloader и предназначен для автоматического обновления подов при изменении ConfigMap и Secret. Для управления используются аннотации. Модуль запускается на **системных** узлах.

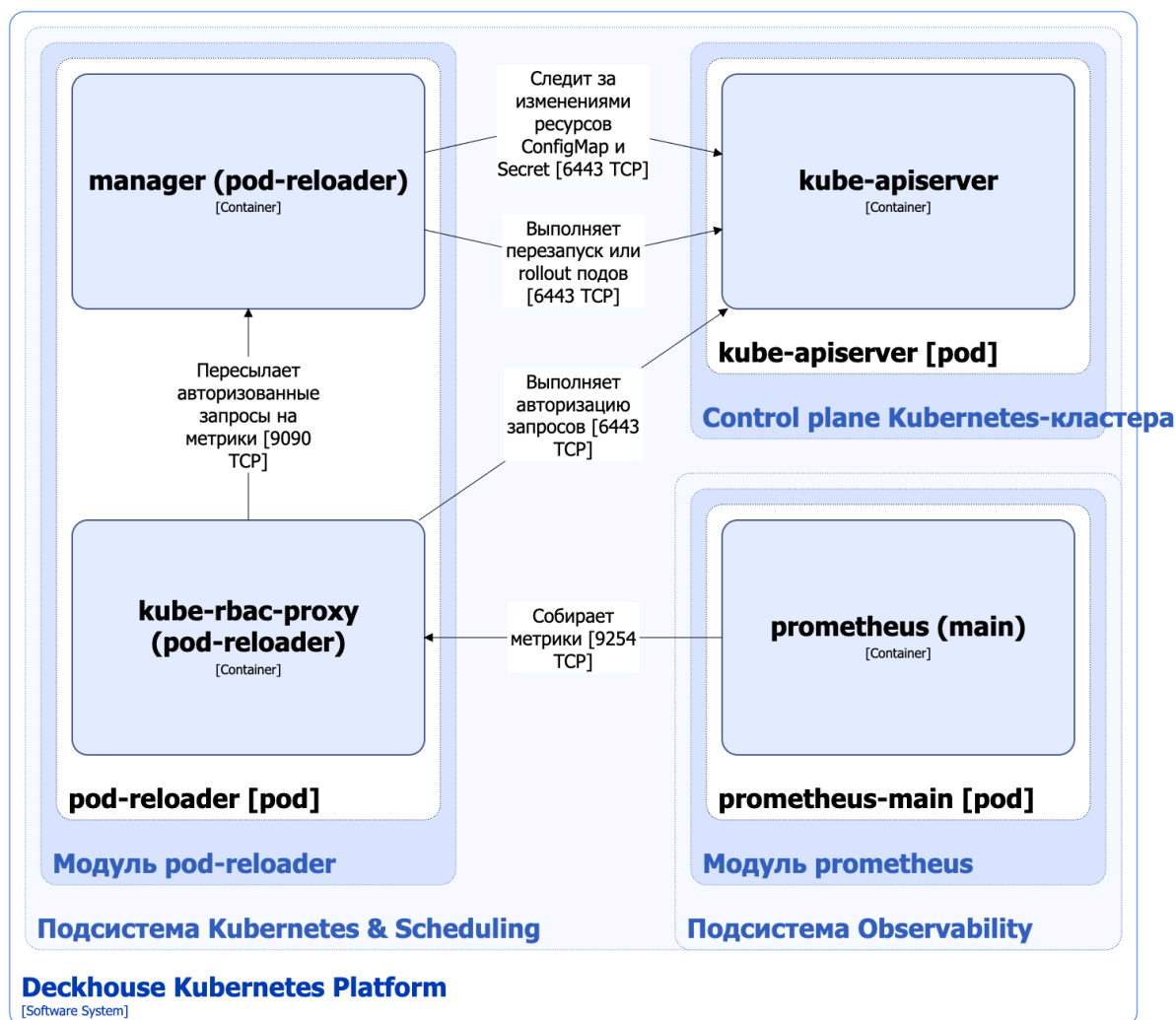
Подробнее о настройках модуля и примерах его использования — в документации модуля.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `pod-reloader` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP показаны на следующей диаграмме:



Компоненты модуля

Модуль `pod-reloader` состоит из одного компонента `pod-reloader`, включающего следующие контейнеры:

- **manager** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищённого доступа к метрикам `pod-reloader`. Является Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует с компонентом `kube-apiserver`:

- следит за изменениями ресурсов ConfigMap и Secret;
- выполняет перезапуск или обновление (rollout) подов;
- выполняет авторизацию запросов на получение метрик.

С модулем взаимодействует компонент `prometheus-main` — собирает метрики модуля.

Дополнительные ресурсы

- Документация модуля `pod-reloader`

Подсистема Cluster & Infrastructure

Данный раздел посвящён архитектуре подсистемы Cluster & Infrastructure платформы Deckhouse Kubernetes Platform (DKP).

Подсистема Cluster & Infrastructure отвечает за инфраструктурную часть управления Kubernetes-кластером. Управление узлами кластера реализовано с помощью модуля `node-manager`, а взаимодействие с IaaS-провайдерами — через соответствующие модули семейства `cloud-provider-`.

В разделе описаны:

- Механизмы управления всеми используемыми в DKP типами узлов, а также гибридными группами узлов и кластерами.
- Служба `bashible`, которая является ключевым компонентом подсистемы Cluster & Infrastructure. `Bashible` используется модулем `node-manager` для управления конфигурацией узлов.

В подсистему Cluster & Infrastructure также входят следующие модули:

- `chrony` — обеспечивает синхронизацию времени на всех узлах кластера;
- `registry-packages-proxy` — предоставляет внутренний прокси-сервер для пакетов хранилища образов контейнеров;
- `terraform-manager` — предоставляет инструменты для работы с состоянием Terraform в Kubernetes-кластере;
- модули облачных провайдеров, поддерживаемых DKP:
 - `cloud-provider-aws`;
 - `cloud-provider-azure`;
 - `cloud-provider-dvp`;
 - `cloud-provider-dynamix`;
 - `cloud-provider-gcp`;
 - `cloud-provider-huaweicloud`;
 - `cloud-provider-openstack`;
 - `cloud-provider-vcd`;
 - `cloud-provider-vmware`;
 - `cloud-provider-yandex`;
 - `cloud-provider-zvirt`.

Управление узлами

Типы узлов и механика добавления

В Deckhouse Kubernetes Platform (DKP) узлы разделяются на следующие типы:

- **CloudEphemeral** — узлы автоматически заказываются, создаются и удаляются в настроенном облачном провайдере.
- **CloudPermanent** — постоянные узлы, создаваемые и обновляемые модулем `node-manager`. Узлы отличаются тем, что их конфигурация берется не из кастомного ресурса `NodeGroup`, а из специального ресурса `<PROVIDER>ClusterConfiguration` (например, `AWSClusterConfiguration` для AWS). Другое важное отличие в том, что для применения конфигурации таких узлов необходимо выполнить `dhctl converge` (запустив инсталлятор DKP). Примером `CloudPermanent`-узла облачного кластера является `master`-узел кластера.
- **CloudStatic** — создаются вручную или любыми внешними инструментами, размещается в том же облаке, с которым настроена интеграция у одного из облачных провайдеров:
 - Узлы типа `CloudStatic` обладают рядом особенностей, связанных с интеграцией с облачным провайдером. Такие узлы управляются компонентом `cloud-controller-manager`, в результате чего:
 - в объект `Node` автоматически добавляются метаданные о зоне и регионе размещения;
 - при удалении виртуальной машины из облака, соответствующий объект `Node` также будет удалён из кластера;
 - доступна работа CSI-драйвера для подключения облачных дисков.
- **Static** — статический узел, размещенный на сервере `bare metal` или виртуальной машине. В случае облачной инфраструктуры такой узел не управляется компонентом `cloud-controller-manager`, даже если включен один из облачных провайдеров. Подробнее про работу со статическими узлами можно прочитать в документации модуля `node-manager`.

Узлы добавляются в кластер путём создания объекта `NodeGroup`, который описывает тип, параметры и конфигурацию группы узлов. В случае `CloudEphemeral`-групп DKP интерпретирует этот объект и автоматически создаёт соответствующие узлы, регистрируя их в Kubernetes-кластере. Для других типов `NodeGroup` (например, `CloudPermanent` или `Static`) создание и регистрация узлов должны быть выполнены вручную или внешними инструментами.

Также поддерживается сценарий гибридных групп, где одна `NodeGroup` может включать как развернутые в облаке узлы типа `Static`, так и статические узлы (серверы `bare-metal`, виртуальные

машины). Например, основную нагрузку могут нести серверы bare-metal, а облачные инстансы использоваться как масштабируемое дополнение при пиковых нагрузках.

Автоматическое развертывание, настройка и обновление узлов Kubernetes

Автоматическое развертывание (в *static/hybrid* — частично), настройка и дальнейшее обновление ПО работают на любых кластерах, независимо от его размещения в облаке или на bare metal.

Развертывание узлов Kubernetes

DKP автоматически разворачивает узлы кластера, выполняя следующие **идемпотентные** операции:

- Настройку и оптимизацию операционной системы для работы с `containerd` и Kubernetes:
 - устанавливаются требуемые пакеты из репозитория дистрибутива;
 - настраиваются параметры работы ядра, параметры журналирования, ротация журналов и другие параметры системы.
- Установку требуемых версий `containerd` и `kubelet`, включение узла в кластер Kubernetes.
- Настройку `kubernetes-api-proxy` и обновление списка `upstream` для балансировки запросов от узла к Kubernetes API.

Поддержка актуального состояния узлов

Для поддержания узлов кластера в актуальном состоянии могут применяться два типа обновлений:

- **Обычные** — такие обновления всегда применяются автоматически и не приводят к остановке или перезагрузке узла.
- **Требующие прерывания** (disruption). Пример таких обновлений — обновление версии ядра или `containerd`, значительная смена версии `kubelet` и т. д. Для этого типа обновлений можно выбрать ручной или автоматический режим (секция параметров `disruptions`). В автоматическом режиме перед обновлением выполняется корректная приостановка работы узла (`drain`) и только после этого производится обновление.

В один момент времени производится обновление только одного узла из группы и только в том случае, когда все узлы группы доступны.

DKP имеет набор встроенных метрик мониторинга, которые позволяют контролировать прогресс обновления, получать уведомления о возникающих во время обновления проблемах или о необходимости получения разрешения на обновление (ручное подтверждение обновления).

Работа с узлами в поддерживаемых облаках

У каждого поддерживаемого облачного провайдера существует возможность автоматического заказа узлов. Для этого необходимо указать требуемые параметры для каждого узла или группы узлов.

В зависимости от провайдера этими параметрами могут быть:

- тип узлов или количество ядер процессора и объем оперативной памяти;
- размер диска;
- настройки безопасности;
- подключаемые сети и др.

Создание, запуск и подключение виртуальных машин к кластеру выполняются автоматически.

Масштабирование узлов в облаке

Возможны два режима масштабирования узлов в группе:

- **Автоматическое масштабирование.**

При дефиците ресурсов, наличии подов в состоянии `Pending`, в группу будут добавлены узлы. При отсутствии нагрузки на один или несколько узлов, они будут удалены из кластера. При работе автомасштабирования учитывается приоритет группы (в первую очередь будет масштабироваться группа, у которой приоритет больше).

Чтобы включить автоматическое масштабирование узлов, необходимо указать разные ненулевые значения минимального (`minPerZone`) и максимального (`maxPerZone`) количества узлов в группе.

- **Фиксированное количество узлов.**

В этом случае DKP будет поддерживать указанное количество узлов (например, заказывая новые в случае выхода из строя старых узлов).

Чтобы указать фиксированное количество узлов в группе и отключить автоматическое

масштабирование, необходимо указать одинаковые значения параметров `minPerZone` и `maxPerZone`.

Работа со статическими узлами

При работе со статическими узлами функции модуля `node-manager` выполняются со следующими ограничениями:

- **Отсутствует заказ узлов.** Непосредственное выделение ресурсов (серверов `bare-metal`, виртуальных машин, связанных ресурсов) выполняется вручную. Дальнейшая настройка ресурсов (подключение узла к кластеру, настройка мониторинга и т.п.) выполняются полностью автоматически (аналогично узлам в облаке) или частично.
- **Отсутствует автоматическое масштабирование узлов.** Доступно поддержание в группе указанного количества узлов при использовании `Cluster API Provider Static` (параметр `staticInstances.count`). Т.е. DKP будет пытаться поддерживать указанное количество узлов в группе, очищая лишние узлы и настраивая новые при необходимости (выбирая их из ресурсов `StaticInstance`, находящихся в состоянии `Pending`).

Настройка/очистка узла, его подключение к кластеру и отключение могут выполняться следующими способами:

- **Вручную**, с помощью подготовленных скриптов.

Для настройки сервера (BM) и ввода узла в кластер нужно загрузить и выполнить специальный `bootstrap`-скрипт. Такой скрипт генерируется для каждой группы статических узлов (каждого ресурса `NodeGroup`). Он находится в секрете `d8-cloud-instance-manager/manual-bootstrap-for-ИМЯ-NODEGROUP`.

Для отключения узла кластера и очистки сервера (виртуальной машины) нужно выполнить скрипт `/var/lib/bashible/cleanup_static_node.sh`, который уже находится на каждом статическом узле.

- **Автоматически**, с помощью `Cluster API Provider Static`.

`Cluster API Provider Static` (CAPS) подключается к серверу (BM) используя ресурсы `StaticInstance` и `SSHCredentials`, выполняет настройку, и вводит узел в кластер.

При необходимости (например, если удален соответствующий серверу ресурс `StaticInstance` или уменьшено количество узлов группы), `Cluster API Provider Static` подключается к узлу кластера, очищает его и отключает от кластера.

- **Вручную с последующей передачей узла под автоматическое управление** `Cluster API Provider Static`.

i Функциональность доступна начиная с версии DKP 1.63.

Для передачи существующего узла кластера под управление CAPS необходимо подготовить для этого узла ресурсы `StaticInstance` и `SSHCredentials` как при автоматическом управлении в пункте выше, однако ресурс `StaticInstance` должен дополнительно быть помечен аннотацией `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Группировка узлов и управление группами

Группировка и управление узлами как связанной группой означает, что все узлы группы будут иметь одинаковые метаданные, взятые из кастомного ресурса `NodeGroup`.

Для групп узлов доступен мониторинг:

- с группировкой параметров узлов на графиках группы;
- с группировкой алертов о недоступности узлов;
- с алертами о недоступности N узлов или N% узлов группы и т. п.

Что такое ресурс Instance

Ресурс `Instance` в Kubernetes представляет собой описание объекта эфемерной виртуальной машины, но без конкретной реализации. Это абстракция, которая используется для управления машинами, созданными с помощью таких инструментов, как `MachineControllerManager` или `Cluster API Provider Static`.

Объект не содержит спецификации. Статус содержит:

1. Ссылку на `InstanceClass`, если он существует для данной реализации.
2. Ссылку на объект `Node` Kubernetes.
3. Текущий статус машины.
4. Информацию о том, как проверить логи создания машины (появляется на этапе создания)

машины).

При создании или удалении машины создается или удаляется соответствующий объект Instance. Самостоятельно ресурс Instance создать нельзя, но можно удалить. В таком случае машина будет удалена из кластера (процесс удаления зависит от деталей реализации).

Когда требуется перезагрузка узлов

Некоторые операции по изменению конфигурации узлов могут потребовать перезагрузки.

Перезагрузка узла может потребоваться при изменении некоторых настроек `sysctl`, например, при изменении параметра `kernel.yama.ptrace_scope` (изменяется при использовании команды `astra-pttrace-lock enable/disable` в Astra Linux).

Влияние параметров NodeGroup на обновление и перезапуск узлов

Параметр NG	Disruption update	Перезаказ узлов	Рестарт kubelet
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+
kubernetesVersion	-	-	+
nodeTemplate	-	-	-
static	-	-	+
update.maxConcurrent	-	-	-

Подробно о всех параметрах можно прочитать в описании кастомного ресурса NodeGroup.

В случае изменения параметров InstanceClass или `instancePrefix` в конфигурации DKP не будет происходить RollingUpdate. DKP создаст новые MachineDeployment, а старые удалит. Количество заказываемых одновременно MachineDeployment определяется параметром `cloudInstances.maxSurgePerZone`.

При обновлении, которое требует прерывания работы узла (disruption update), выполняется процесс вытеснения подов с узла. Если какой-либо под не может быть вытеснен, попытка повторяется каждые 20 секунд до достижения глобального таймаута в 5 минут. После истечения этого времени, поды, которые не удалось вытеснить, удаляются принудительно.

Дополнительные ресурсы

- Настройки провайдера модуля `cloud-provider-aws`
- Документация модуля `node-manager`

Bashible

Bashible-скрипты и служба bashible

Функции управления узлами реализуются с помощью специально подготовленных bash-скриптов, называемых **bashible**. Так же называется служба, которая работает на узлах кластера и используется для запуска данных скриптов. Набор скриптов называется бандлом (bundle).

Используются 4 бандла:

- скрипты установки bashible;
- скрипты бутстрапа первого узла;
- скрипты настройки узла для определенного облачного провайдера (например, AWS);

- основные (common) скрипты.

Скрипты представляют собой *gotemplate*-шаблоны, что позволяет гибко настраивать узел в зависимости от группы. Сами скрипты должны быть написаны так, чтобы они могли корректно выполняться при повторном запуске в случае ошибки и при повторном прогоне. Отдельный скрипт называется степом (шагом).

Основные этапы настройки узла:

- Настройка NodeUser для обеспечения доступа к узлу.
- Установка CA-сертификатов.
- Создание и добавление в `PATH` каталога `/opt/deckhouse/bin`, в котором хранятся бинарные файлы.
- Скачивание необходимых пакетов из `registrypackages`.
- Установка и настройка CRI containerd.
- Скачивание и настройка **kubernetes-api-proxy**. Компонент отвечает за доступ к API Kubernetes, представляет собой NGINX с набором upstream-серверов к master-узлам. Это обеспечивает HA-доступ к API на случай, если один master-узел недоступен, а также балансировку нагрузки к API.
- Установка, настройка и запуск kubelet.
- Запуск службы bashible, которая выполняет `bashible.sh` каждую минуту.
- Перезагрузка узла при необходимости.

Bashible-api-server

Учитывая большое количество модификаций bashible-скриптов для разных поддерживаемых ОС, хранить все варианты в базе etcd невозможно из-за ограничения на размер ключа, а также избыточной нагрузки на etcd. По этой причине был разработан компонент **bashible-api-server**, который генерирует bashible-скрипты из шаблонов, хранящихся в кастомных ресурсах.

Bashible-api-server представляет собой Kubernetes Extension API Server, который разворачивается на master-узлах.

При обращении к kube-apiserver за ресурсами, содержащими бандлы bashible, kube-apiserver перенаправляет запрос в bashible-api-server и возвращает сформированный результат. Взаимодействия bashible и bashible-api-server показаны на схемах архитектуры модуля `node-manager` (например, на схеме для CloudEphemeral-узлов).

Bashible-api-server возвращает следующие ресурсы:

- **bootstrap-скрипт** второй фазы, который загружается из первой фазы;
- **bashibles** — скрипт `bashible.sh`;
- **nodegroupbundles** — в нем рендерится бандл, включающий набор скриптов для бутстрапа и настройки узла.

Все эти ресурсы можно получить как через API, так и с помощью команды `kubectl`, указав имя группы узлов:

- `kubectl get bootstrap.bashible.deckhouse.io master -o yaml`;
- `kubectl get bashibles.bashible.deckhouse.io master -o yaml`;
- `kubectl get nodegroupbundles.bashible.deckhouse.io master -o yaml`.

Также bashible-api-server вычисляет контрольную сумму всех скриптов группы узлов. Это необходимо для реализации механизма обновления и корректного обновления статуса группы. Контрольные суммы записываются в секрет `d8-cloud-instance-manager/configuration-checksums`. Изменение контрольной суммы инициирует перезапуск bashible-скриптов на узлах при изменении конфигурации. Кроме того, контрольная сумма службы bashible сбрасывается каждые 4 часа для принудительного перезапуска bashible.

Модуль node-manager

Управление узлами кластера осуществляется с помощью модуля `node-manager`.

Подробнее с описанием функций и настроек модуля, а также примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

В зависимости от типа узлов архитектура модуля отличается по составу компонентов. На следующих страницах описана архитектура модуля для различных типов узлов:

- Управление CloudEphemeral-узлами
- Управление CloudPermanent-узлами
- Управление CloudStatic-узлами
- Управление Static-узлами
- Управление гибридными группами и кластерами

Дополнительные ресурсы

- Документация модуля `node-manager`

Управление CloudEphemeral-узлами

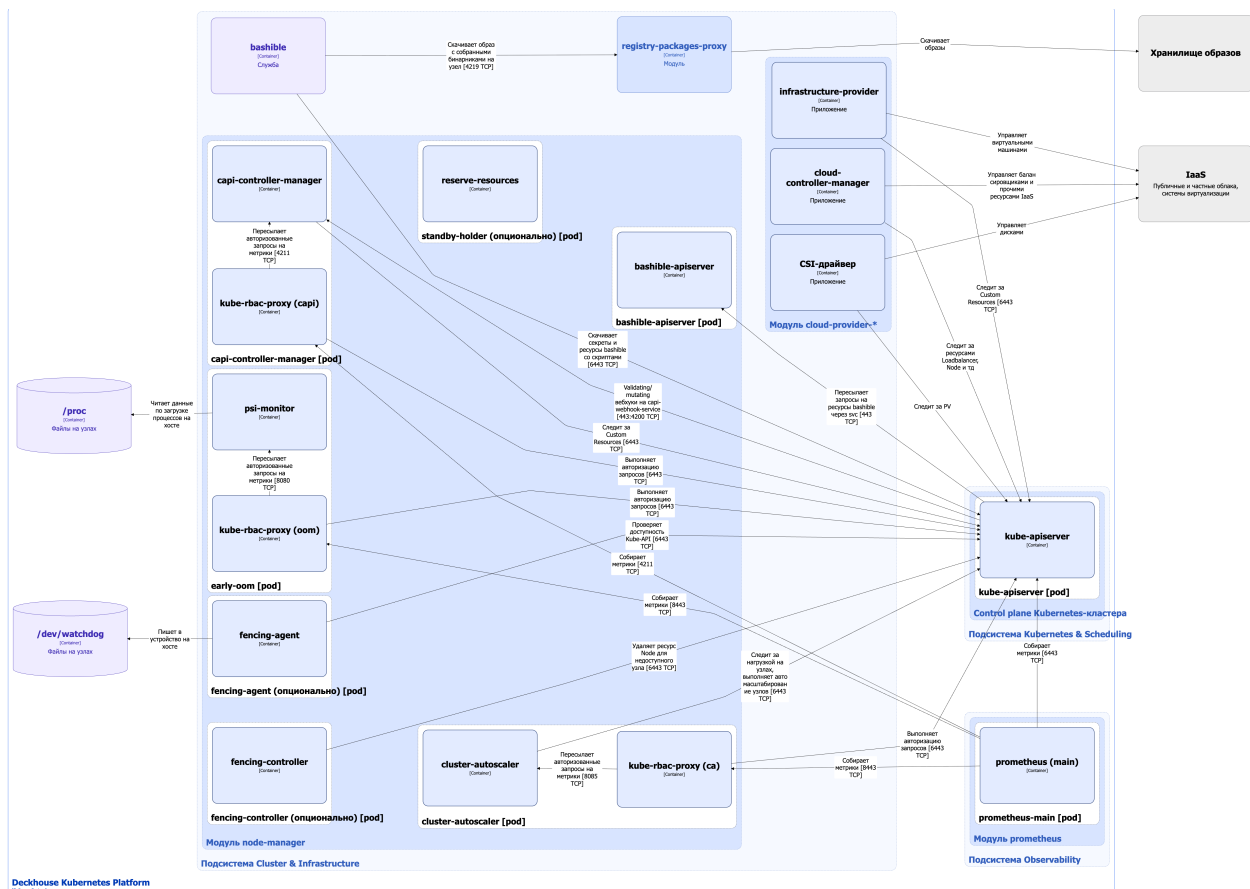
На данной странице описана архитектура модуля `node-manager` для CloudEphemeral-узлов.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

i Bashible — это ключевой компонент подсистемы Cluster & Infrastructure, обеспечивающий работу модуля `node-manager`. При этом он не является компонентом модуля, поскольку работает на уровне ОС в качестве системной службы. Bashible подробно описан в соответствующем разделе документации

Модуль, управляющий CloudEphemeral-узлами, состоит из следующих компонентов:

1. **Bashible-api-server** — Kubernetes Extension API Server, развернутый на master-узлах. Генерирует bashible-скрипты из шаблонов, хранящихся в кастомных ресурсах. При обращении к `kube-apiserver` за ресурсами, содержащими бандлы `bashible`, `kube-apiserver` перенаправляет запрос в `bashible-api-server` и возвращает сформированный результат. Подробнее с описанием работы `bashible` и `bashible-api-server` можно ознакомиться в соответствующем разделе документации.
2. **Capi-controller-manager** (Deployment) — основные контроллеры из проекта Kubernetes Cluster

API. Cluster API является расширением Kubernetes, которое дает возможность управлять кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Под capi-controller-manager состоит из следующих контейнеров:

- **control-plane-manager** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера.

3. **Cluster-autoscaler** (Deployment) — дополнительный компонент Kubernetes, который автоматически изменяет количество узлов в кластере в зависимости от нагрузки. Подробнее с автоматическим масштабированием узлов можно ознакомиться в разделе документации по управлению узлами.

Компонент включает в себя следующие контейнеры:

- **cluster-autoscaler** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам **cluster-autoscaler**.

4. **Early-oom** (DaemonSet) — на каждом узле разворачивается под, который считывает из каталога `/proc` метрики по загрузке ресурсов на хосте и в случае повышенной нагрузки завершает поды раньше, чем это сделает kubelet. **Early-oom** по умолчанию включен, но его можно отключить в настройках модуля в случае, если он создаёт проблемы для нормальной работы узлов.

Включает в себя следующие контейнеры:

- **psi-monitor** — основной контейнер, который отслеживает метрику *PSI (Pressure Stall Information)*, отражающую время, в течение которого процессы ожидают освобождения определённых ресурсов, таких как CPU, память или I/O;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам **early-oom**.

5. **Fencing-agent** (DaemonSet) и **fencing-controller** — компоненты, реализующие механизм fencing. Принцип работы компонентов подробно разобран в описании параметра `spec.fencing.mode` ресурса NodeGroup. Подробнее о том, как механизм fencing обрабатывает разные типы узлов, можно почитать в разделе «FAQ» документации модуля `node-manager`.

6. **Standby-holder** (Deployment) — под для резервирования узлов. При включенном параметре `spec.cloudinstances.standby` кастомного ресурса NodeGroup в соответствующей группе узлов во всех зонах создаются резервные узлы.

Резервный узел — это узел кластера, на котором резервируются ресурсы, доступные в любой момент для масштабирования. Наличие такого узла позволяет cluster-autoscaler не ждать инициализации узла (которая может занимать несколько минут), а сразу размещать на нем нагрузку.

Standby-holder не выполняет никакой полезной работы, а резервирует ресурсы, не позволяя cluster-autoscaler удалить временно неиспользуемый узел.

У пода standby-holder минимальный PriorityClass, и он вытесняется с узла при появлении реальной нагрузки. Подробнее о приоритизации и вытеснении подов можно почитать в документации Kubernetes.

Под содержит один контейнер **reserve-resources**.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- получение секрета `kube-system/d8-node-manager-cloud-provider` для подключения к облаку;
- работа с кастомными ресурсами Cluster API;
- работа с ресурсами Node;
- отслеживание нагрузки на узлах;
- автомасштабирование узлов;
- авторизация запросов на метрики.

2. Файлы на узлах:

- `/proc` — читает метрики PSI для OOM Kill;
- `/dev/watchdog` — отправляет сигнал в Watchdog для сброса сторожевого таймера.

i Модуль взаимодействует с модулем `cloud-provider` через `kube-apiserver`, используя секрет `kube-system/d8-node-manager-cloud-provider`, для получения всех необходимых настроек подключения к облаку и создания `CloudEphemeral`-узлов. Также `cloud-provider` предоставляет модулю `node-manager` шаблоны для создания кастомных ресурсов `Cluster API`, специфичных для определенных провайдеров.

С модулем взаимодействуют следующие внешние для него компоненты:

1. **Kube-apiserver:**

- выполняет `mutating`- и `validating`-вебхуки `capi-controller-manager`;
- пересылает в `bashible-api-server` запросы на ресурсы `bashible`.

2. **Prometheus-main** — сбор метрик компонентов модуля `node-manager`.

Особенности архитектуры, специфичные для CloudEphemeral-узлов

1. Узлы эфемерны, автоматически создаются и удаляются модулем.
2. Для взаимодействия с инфраструктурой облака необходим установленный и настроенный облачный провайдер (`cloud-provider-*` на схеме). Включает также `csi-driver` и `cloud-controller-manager`.
3. **Capi-controller-manager** — компонент, обеспечивающий жизненный цикл самого кластера и его узлов. Не заказывает узлы в облаке самостоятельно, работает с кастомными ресурсами более высокого уровня, не привязанного к инфраструктуре. Генерирует инфраструктурные кастомные ресурсы, оставляя всю работу для инфраструктурного провайдера, который разворачивается модулем конкретного облачного провайдера `cloud-provider`.
4. **Cluster-autoscaler** — обеспечивает автомасштабирование узлов кластера.
5. Поддерживается резервирование узлов.

Дополнительные ресурсы

- Документация модуля `node-manager`

Управление CloudPermanent-узлами

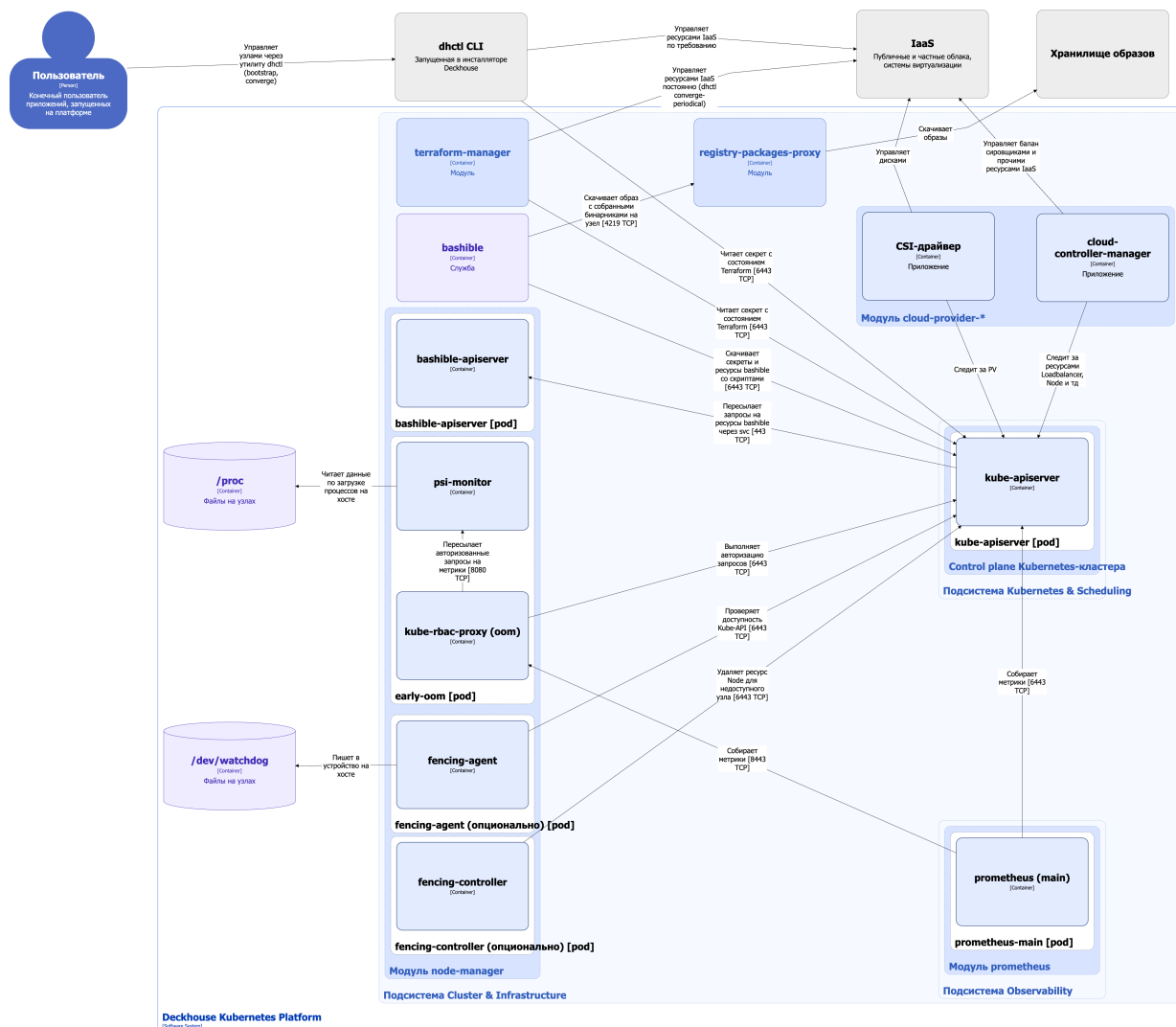
На данной странице описана архитектура модуля `node-manager` для `CloudPermanent`-узлов.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы `Kubernetes` (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-manager` на уровне 2 модели `C4` и его взаимодействия с другими компонентами `Deckhouse Kubernetes Platform (DKP)` изображены на следующей диаграмме:



Компоненты модуля

i Bashible — это ключевой компонент подсистемы Cluster & Infrastructure, обеспечивающий работу модуля node-manager. При этом он не является компонентом модуля, поскольку работает на уровне ОС в качестве системной службы. Bashible подробно описан в соответствующем разделе документации

Модуль, управляющий CloudPermanent-узлами, состоит из следующих компонентов:

- Bashible-api-server** — Kubernetes Extension API Server, развернутый на master-узлах. Генерирует bashible-скрипты из шаблонов, хранящихся в кастомных ресурсах. При обращении к kube-apiserver за ресурсами, содержащими бандлы bashible, kube-apiserver перенаправляет запрос в bashible-api-server и возвращает сформированный результат. Подробнее с описанием работы bashible и bashible-api-server можно ознакомиться в соответствующем разделе документации.
- Early-oom** (DaemonSet) — на каждом узле разворачивается под, который считывает из каталога `/proc` метрики по загрузке ресурсов на хосте и в случае повышенной нагрузки завершает поды раньше, чем это сделает kubelet. **Early-oom** по умолчанию включен, но его можно отключить в настройках модуля в случае, если он создаёт проблемы для нормальной работы узлов.

Включает в себя следующие контейнеры:

- psi-monitor** — основной контейнер, который отслеживает метрику *PSI (Pressure Stall Information)*, отражающую время, в течение которого процессы ожидают освобождения определённых ресурсов, таких как CPU, память или I/O;
- kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам **early-oom**.

- Fencing-agent** (DaemonSet) и **fencing-controller** — компоненты, реализующие механизм fencing. Принцип работы компонентов подробно разобран в описании параметра `спес.fencing.mode` ресурса NodeGroup. Подробнее о том, как механизм fencing обрабатывает разные типы узлов, можно почитать в разделе «FAQ» документации модуля node-manager.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- работа с ресурсами Node;
- авторизация запросов на метрики.

2. Файлы на узлах:

- `/proc` - читает метрики PSI для OOM Kill.
- `/dev/watchdog` - отправляет сигнал в Watchdog для сброса сторожевого таймера.

С модулем взаимодействуют следующие внешние для него компоненты:

1. Kube-apiserver:

- пересылает в `bashible-api-server` запросы на ресурсы `bashible`.

2. Prometheus-main — сбор метрик компонентов модуля `node-manager`.

Особенности архитектуры, специфичные для CloudPermanent-узлов

1. Узлы постоянны, создаются, управляются и удаляются пользователем. Пользователь управляет узлами не напрямую в инфраструктуре, а через утилиту `dhctl`, запущенную в инсталляторе DKP.
2. `Terraform-manager` — модуль, используемый для автоматического управления ресурсами облачной инфраструктуры. Он проверяет состояние Terraform и применяет неdestructive изменения к инфраструктурным ресурсам. С архитектурой модуля можно ознакомиться на соответствующей странице документации.
3. `Csi-driver` — используется для заказа дисков в облачной инфраструктуре.
4. `Cloud-controller-manager` — используется для заказа балансировщиков и прочих инфраструктурных ресурсов согласно своей спецификации.
5. `Infrastructure-provider` — не требуется, вся работа с узлами осуществляется пользователем через утилиту `dhctl` и модуль `terraform-manager`.
6. Автоматическое масштабирование узлов не поддерживается.

Дополнительные ресурсы

- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Управление CloudStatic-узлами

На данной странице описана архитектура модуля `node-manager` для CloudStatic-узлов.

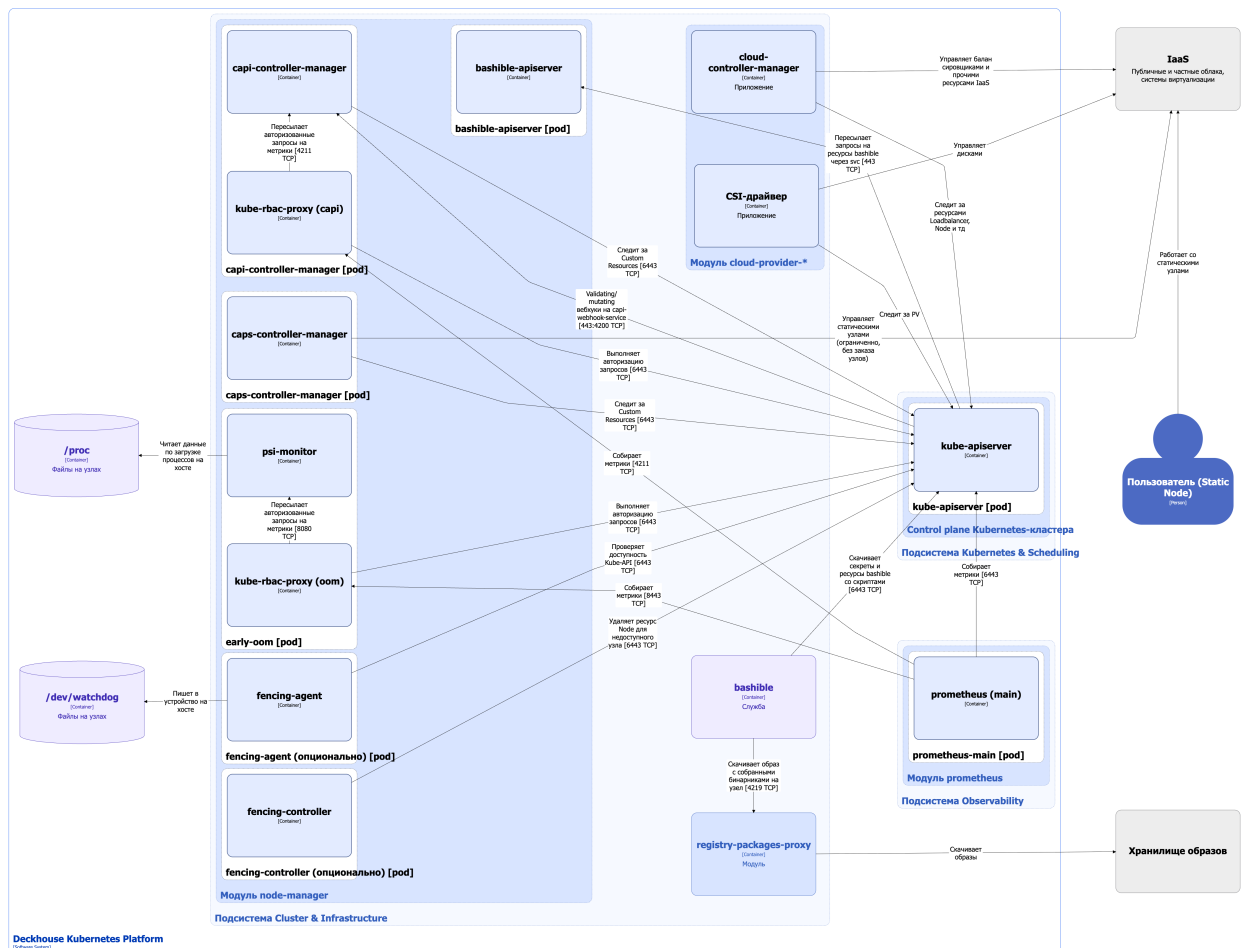
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

i Bashible — это ключевой компонент подсистемы Cluster & Infrastructure, обеспечивающий работу модуля node-manager. При этом он не является компонентом модуля, поскольку работает на уровне ОС в качестве системной службы. Bashible подробно описан в соответствующем разделе документации

Модуль, управляющий CloudStatic-узлами, состоит из следующих компонентов:

1. **Bashible-api-server** — Kubernetes Extension API Server, развернутый на master-узлах. Генерирует bashible-скрипты из шаблонов, хранящихся в кастомных ресурсах. При обращении к kube-apiserver за ресурсами, содержащими бандлы bashible, kube-apiserver перенаправляет запрос в bashible-api-server и возвращает сформированный результат. Подробнее с описанием работы bashible и bashible-api-server можно ознакомиться в соответствующем разделе документации.
2. **Capi-controller-manager** (Deployment) — основные контроллеры из проекта Kubernetes Cluster API. Cluster API является расширением Kubernetes, которое дает возможность управлять кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Под capi-controller-manager состоит из следующих контейнеров:
 - **control-plane-manager** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера.
3. **Caps-controller-manager** (Deployment) — CAPI Provider Static (CAPS), реализация провайдера декларативного управления статическими узлами (серверами bare metal или виртуальными машинами) для проекта Cluster API Kubernetes. Работает как дополнение к capi-controller-manager.

CAPS представляет собой дополнительный слой абстракции над существующим функционалом DKP по автоматической настройке и очистке статических узлов с помощью скриптов, генерируемых для каждой группы узлов. Компонент не привязан к конкретному облаку. Подробнее про работу CAPS можно почитать в документации модуля `node-manager`.
4. **Early-oom** (DaemonSet) — на каждом узле разворачивается под, который считывает из каталога `/proc` метрики по загрузке ресурсов на хосте и в случае повышенной нагрузки завершает поды раньше, чем это сделает kubelet. **Early-oom** по умолчанию включен, но его можно отключить в

настройках модуля в случае, если он создаёт проблемы для нормальной работы узлов.

Включает в себя следующие контейнеры:

- **psi-monitor** — основной контейнер, который отслеживает метрику *PSI (Pressure Stall Information)*, отражающую время, в течение которого процессы ожидают освобождения определённых ресурсов, таких как CPU, память или I/O;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам **early-oom**.

5. **Fencing-agent** (DaemonSet) и **fencing-controller** — компоненты, реализующие механизм fencing. Принцип работы компонентов подробно разобран в описании параметра `спес.fencing.mode` ресурса NodeGroup. Подробнее о том, как механизм fencing обрабатывает разные типы узлов, можно почитать в разделе «FAQ» документации модуля `node-manager`.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- работа с кастомными ресурсами Cluster API;
- работа с ресурсами Node;
- авторизация запросов на метрики.

2. Файлы на узлах:

- `/proc` — читает метрики PSI для OOM Kill;
- `/dev/watchdog` — отправляет сигнал в Watchdog для сброса сторожевого таймера.

3. Инфраструктура:

- управление статическими узлами (ограниченно, без заказа узлов).

С модулем взаимодействуют следующие внешние для него компоненты:

1. Kube-apiserver:

- выполняет mutating- и validating-вебхуки capi-controller-manager;
- пересылает в bashible-api-server запросы на ресурсы bashible.

2. Prometheus-main — сбор метрик компонентов модуля node-manager.

Особенности архитектуры, специфичные для CloudStatic-узлов

1. Пользователь создает и настраивает узлы следующими способами:

- вручную — с помощью подготовленных платформой bashible-скриптов;
- вручную с последующей передачей узла под автоматическое управление CAPS;
- автоматически, с помощью CAPS.

2. Capi-controller-manager — компонент, обеспечивающий жизненный цикл самого кластера и его узлов. Не заказывает узлы в облаке самостоятельно, работает с кастомными ресурсами более высокого уровня, не привязанного к инфраструктуре. Генерирует инфраструктурные кастомные ресурсы, оставляя всю работу для инфраструктурного провайдера (CAPS).

3. Caps-controller-manager — компонент, управляющий статическими узлами (ограниченно, без заказа узлов).

4. Csi-driver — используется для заказа дисков в облачной инфраструктуре.

5. Cloud-controller-manager — используется для заказа балансировщиков и прочих инфраструктурных ресурсов согласно своей спецификации.

6. Infrastructure-provider конкретного облака не требуется, его роль выполняет caps-controller-manager.

7. Автоматическое масштабирование узлов не поддерживается.

Дополнительные ресурсы

- Документация модуля `node-manager`

Управление Static-узлами

На данной странице описана архитектура модуля `node-manager` для Static-узлов.

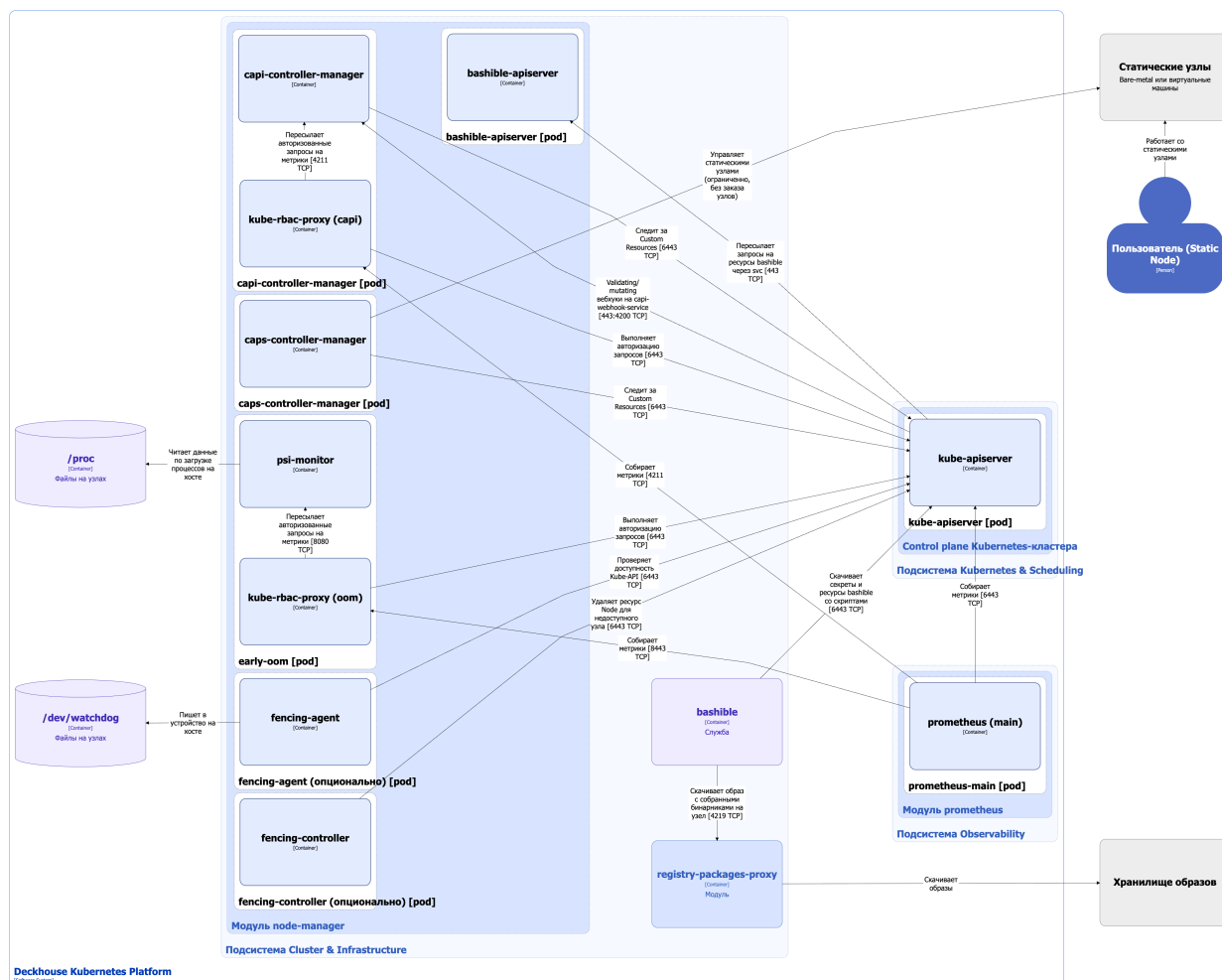
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

i

Bashible — это ключевой компонент подсистемы Cluster & Infrastructure, обеспечивающий работу модуля `node-manager`. При этом он не является компонентом модуля, поскольку работает на уровне ОС в качестве системной службы. Bashible подробно описан в соответствующем разделе документации

Модуль, управляющий Static-узлами, состоит из следующих компонентов:

1. **Bashible-api-server** — Kubernetes Extension API Server, развернутый на master-узлах. Генерирует bashible-скрипты из шаблонов, хранящихся в кастомных ресурсах. При обращении к `kube-apiserver` за ресурсами, содержащими бандлы bashible, `kube-apiserver` перенаправляет запрос в `bashible-api-server` и возвращает сформированный результат. Подробнее с описанием работы bashible и `bashible-api-server` можно ознакомиться в соответствующем разделе документации.
2. **Capi-controller-manager** (Deployment) — основные контроллеры из проекта Kubernetes Cluster API. Cluster API является расширением Kubernetes, которое дает возможность управлять кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Под `capi-controller-manager` состоит из следующих контейнеров:
 - **control-plane-manager** — основной контейнер;

- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера.
3. **Caps-controller-manager** (Deployment) — CAPI Provider Static (CAPS), реализация провайдера декларативного управления статическими узлами (серверами bare metal или виртуальными машинами) для проекта Cluster API Kubernetes. Работает как дополнение к capi-controller-manager.

CAPS представляет собой дополнительный слой абстракции над существующим функционалом DKP по автоматической настройке и очистке статических узлов с помощью скриптов, генерируемых для каждой группы узлов. Компонент не привязан к конкретному облаку. Подробнее про работу CAPS можно почитать в документации модуля `node-manager`.

4. **Early-oom** (DaemonSet) — на каждом узле разворачивается под, который считывает из каталога `/proc` метрики по загрузке ресурсов на хосте и в случае повышенной нагрузки завершает поды раньше, чем это сделает kubelet. **Early-oom** по умолчанию включен, но его можно отключить в настройках модуля в случае, если он создаёт проблемы для нормальной работы узлов.

Включает в себя следующие контейнеры:

- **psi-monitor** — основной контейнер, который отслеживает метрику *PSI (Pressure Stall Information)*, отражающую время, в течение которого процессы ожидают освобождения определённых ресурсов, таких как CPU, память или I/O;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам **early-oom**.
5. **Fencing-agent** (DaemonSet) — разворачивается на группе узлов при включённом параметре `spec.fencing` кастомного ресурса `NodeGroup`.

Принцип работы компонента подробно разобран в описании параметра `spec.fencing.mode` ресурса `NodeGroup`. Подробнее о том, как механизм `fencing` обрабатывает разные типы узлов, можно почитать в разделе «FAQ» документации модуля `node-manager`.

Состоит из одного контейнера:

- **fencing-agent** — выполняет необходимые проверки. Сигнал в `watchdog` отправляется посредством записи в файл `/dev/watchdog` на хосте.
6. **Fencing-controller** — контроллер, который отслеживает все узлы с лейблом `node-manager.deckhouse.io/fencing-enabled`.

Если узел недоступен более 60 секунд, контроллер удаляет с него все поды, но **не удаляет сам объект Node**: для статических узлов (`node.deckhouse.io/type=Static`) регистрация в кластере сохраняется, чтобы узел мог вернуться в работу после ручного восстановления.

Подробнее о том, как механизм `fencing` обрабатывает разные типы узлов, можно почитать в разделе «FAQ» документации модуля `node-manager`.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- работа с кастомными ресурсами Cluster API;
- работа с ресурсами Node;
- авторизация запросов на метрики.

2. Файлы на узлах:

- `/proc` — читает метрики PSI для OOM Kill;
- `/dev/watchdog` — отправляет сигнал в Watchdog для сброса сторожевого таймера.

С модулем взаимодействуют следующие внешние для него компоненты:

1. Kube-apiserver:

- выполняет mutating- и validating-вебхуки capi-controller-manager;
- пересылает в bashible-api-server запросы на ресурсы bashible.

2. Prometheus-main — сбор метрик компонентов модуля node-manager.

Особенности архитектуры, специфичные для Static-узлов

1. Пользователь создает и настраивает узлы следующими способами:

- вручную — с помощью подготовленных платформой bashible-скриптов;
- вручную с последующей передачей узла под автоматическое управление CAPS;
- автоматически, с помощью CAPS.

2. **Capi-controller-manager** — компонент, обеспечивающий жизненный цикл самого кластера и его узлов. Не заказывает узлы в облаке самостоятельно, работает с кастомными ресурсами более высокого уровня, не привязанного к инфраструктуре. Генерирует инфраструктурные кастомные ресурсы, оставляя всю работу для инфраструктурного провайдера, который развертывается модулем конкретного облачного провайдера. Для статических узлов это CAPS.
3. **Caps-controller-manager** — компонент, управляющий статическими узлами (ограниченно, без заказа узлов).
4. Использование Static-узлов возможно не только на bare metal, но и в облаке. В случае облака, такой узел не управляется cloud-controller-manager, даже если включен один из облачных провайдеров. Csi-driver также не устанавливается на такие узлы.
5. Автоматическое масштабирование узлов не поддерживается.

Дополнительные ресурсы

- Документация модуля `node-manager`

Управление гибридными группами узлов и кластерами

Следует различать гибридные группы узлов и гибридные кластеры:

- **Гибридные группы узлов** могут включать как развернутые в облаке Static-узлы, так и серверы, развернутые в собственном ЦОД заказчика (bare metal или виртуальные машины).

Например, основную нагрузку могут нести серверы bare metal, а облачные инстансы использоваться как масштабируемое дополнение при пиковых нагрузках. Так как и в ЦОД заказчика и в облаке используются узлы одного типа (Static), архитектура модуля `node-manager` в этом случае будет соответствовать варианту для Static-узлов. Единственным обязательным требованием является наличие L3-связи между собственным ЦОД и облаком. Например, Yandex Cloud предоставляет механизм Cloud Interconnect.

- **Гибридные кластеры** — это кластеры DKP, в которых одновременно присутствуют:
 - NodeGroup со Static-узлами, развернутыми в собственном ЦОД заказчика (bare metal или виртуальные машины);
 - NodeGroup с CloudEphemeral-узлами, развернутыми в облаке.

В этом случае будут развернуты компоненты, необходимые для управления узлами обоих типов. Архитектура модуля `node-manager` будет соответствовать варианту для Static-узлов. Дополнительно будут развернуты компоненты, необходимые для работы CloudEphemeral-узлов:

- **cloud-provider** — обеспечивает взаимодействие с облачной инфраструктурой. Требуется установленный и настроенный провайдер для соответствующего облака. Включает также csi-driver и cloud-controller-manager;
- **cluster-autoscaler** — обеспечивает автомасштабирование узлов кластера.

Как и для гибридных групп узлов, для работы гибридных кластеров требуется наличие L3-связи между собственным ЦОД и облаком.

Дополнительные ресурсы

- Документация модуля `node-manager`

Модуль cloud-provider-aws

Модуль `cloud-provider-aws` обеспечивает интеграцию с облачными ресурсами Amazon Web Services. Он используется модулем `node-manager` для заказа узлов в соответствии с настройками группы узлов.

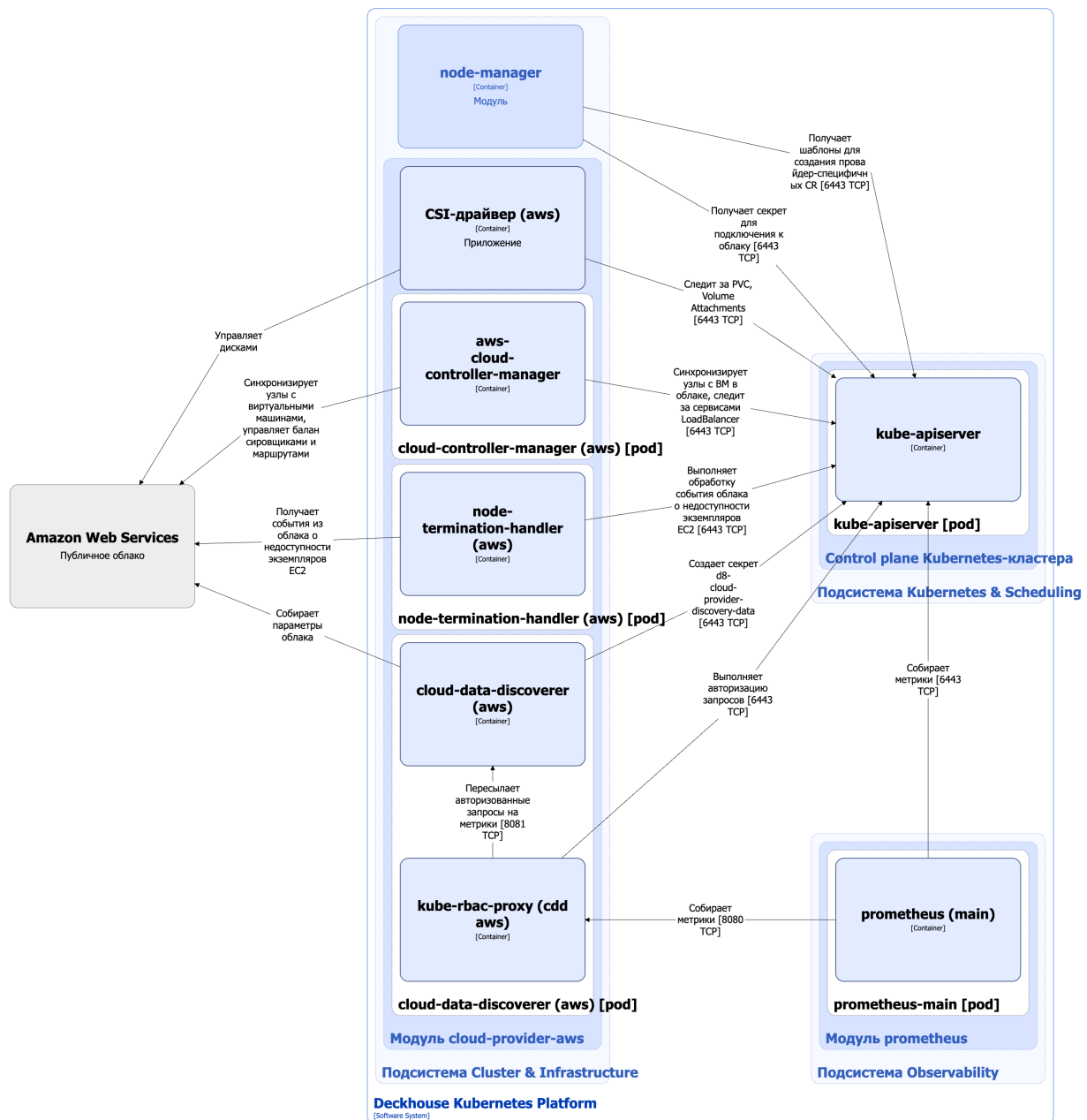
Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-aws` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — реализация cloud controller manager для AWS. Компонент обеспечивает интеграцию с облаком AWS и выполняет следующие функции:
 - о реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resurca Node`;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
 - о при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера;
 - о создает сетевые маршруты для сети `PodNetwork` на стороне AWS.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- о **aws-cloud-controller-manager**.

2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-aws`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.
3. **CSI-драйвер (aws)** — реализация CSI-драйвера для AWS. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться в соответствующем разделе документации.
 4. **Node-termination-handler** — AWS Node Termination Handler, отвечает за обработку DKP событий от сервисов AWS о недоступности экземпляров EC2.

Node-termination-handler обрабатывает следующие события AWS:

- EC2 maintenance events;
- EC2 Spot interruptions;
- ASG Scale-In;
- ASG AZ Rebalance;
- EC2 Instance Termination.

Состоит из одного контейнера:

- **node-termination-handler**.

При получении события, требующего вывода узла из эксплуатации (drain), например при плановом прерывании работы spot-инстанса, node-termination-handler устанавливает на соответствующий узел taint `aws-node-termination-handler/spot-itn`. Модуль `node-manager` обнаруживает этот taint и с помощью хуков модуля выполняет drain, а затем удаляет узел.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. Amazon Web Services:

- получение параметров облака;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- создание сетевых маршрутов для сети `PodNetwork`;
- получение событий о недоступности экземпляров EC2;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-aws` предоставляет модулю `node-manager` следующие артефакты:

- шаблоны для создания кастомных ресурсов для конкретного провайдера, которые `cloud-provider-aws` использует для создания виртуальных машин в облаке;
- секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль `cloud-provider-aws` предоставляет компоненты Terraform/OpenTofu для AWS, которые используются при сборке исполняемого файла утилиты `dhctl` в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- `layouts` — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждая схема включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;

- master-node;
- static-node.

Дополнительные ресурсы

- Документация модуля `cloud-provider-aws`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль `cloud-provider-azure`

Модуль `cloud-provider-azure` обеспечивает интеграцию с облачными ресурсами Microsoft Azure. Он использует модуль `node-manager` для заказа узлов в соответствии с настройками группы узлов.

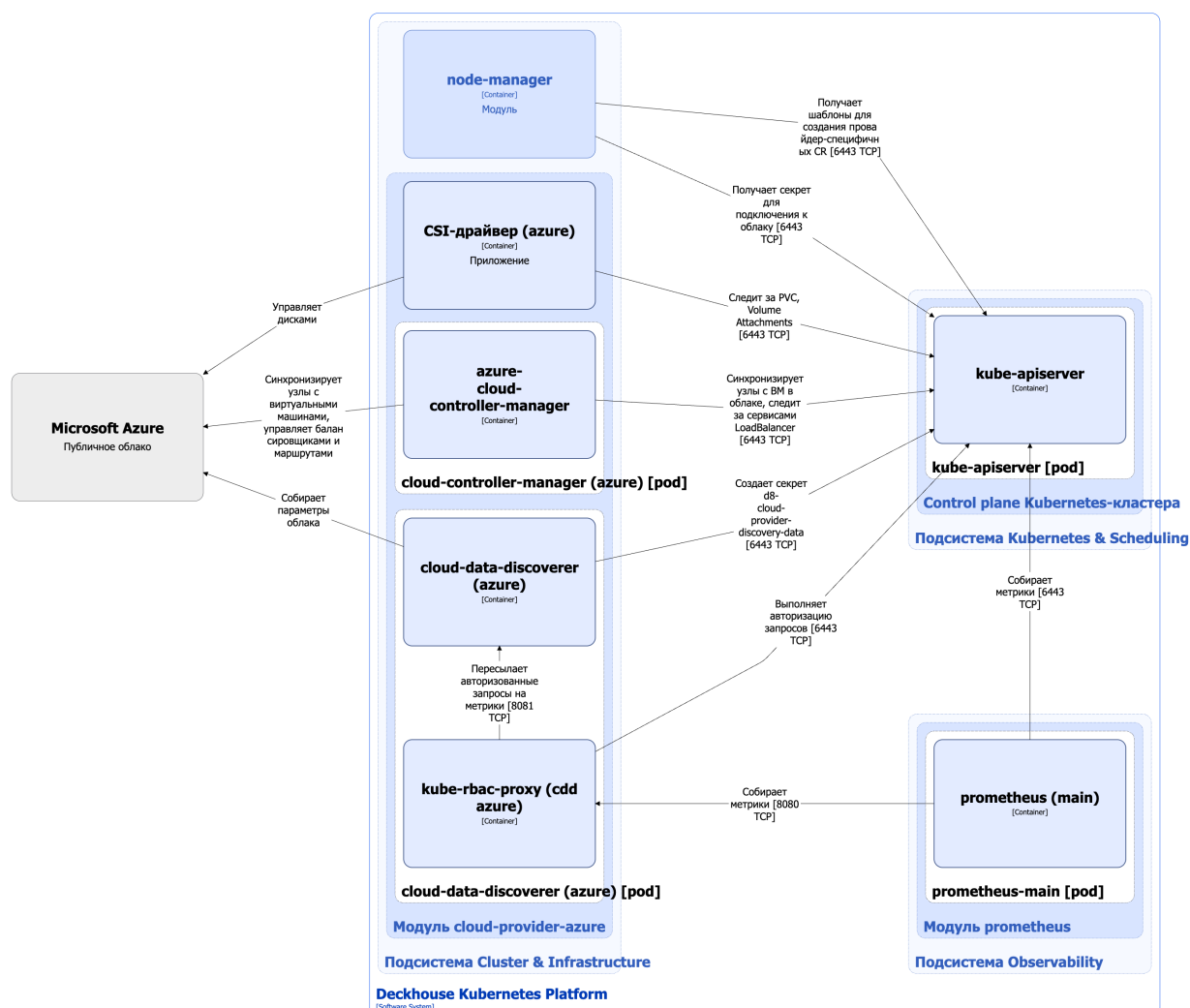
Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-azure` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — реализация cloud controller manager для Microsoft Azure. Компонент обеспечивает интеграцию с облаком Microsoft Azure и выполняет следующие функции:
 - реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resync` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
 - при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера;
 - создает сетевые маршруты для сети `PodNetwork` на стороне Azure.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **azure-cloud-controller-manager**.
2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-azure`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера `cloud-data-discoverer`.
3. **CSI-драйвер (azure)** — реализация CSI-драйвера для Microsoft Azure. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться в соответствующем разделе документации.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
 - создание секрета `kube-system/d8-cloud-provider-discovery-data`;
 - синхронизация узлов Kubernetes с виртуальными машинами в облаке;
 - мониторинг сервисов типа LoadBalancer;
 - авторизация запросов на получение метрик.
2. **Microsoft Azure**:
 - получение параметров облака;
 - получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
 - управление балансировщиками;
 - создание сетевых маршрутов для сети `PodNetwork`;
 - управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик `cloud-data-discoverer`.

Непрямые взаимодействия:

1. Модуль `cloud-provider-azure` предоставляет модулю `node-manager` следующие артефакты:
 - шаблоны для создания кастомных ресурсов для конкретного провайдера, которые `cloud-provider-azure` использует для создания виртуальных машин в облаке;
 - секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.
2. Модуль `cloud-provider-azure` предоставляет компоненты Terraform/OpenTofu для Azure, которые используются при сборке исполняемого файла утилиты `dhctl` в модуле `terraform-manager`, такие как:
 - Terraform/OpenTofu-провайдер;
 - Terraform-модули;

- `layouts` — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждая схема включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-azure`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-dvp

Модуль `cloud-provider-dvp` управляет взаимодействием с облачными ресурсами Deckhouse Virtualization Platform (DVP). Он позволяет модулю `node-manager` использовать ресурсы DVP при заказе узлов для описанной группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

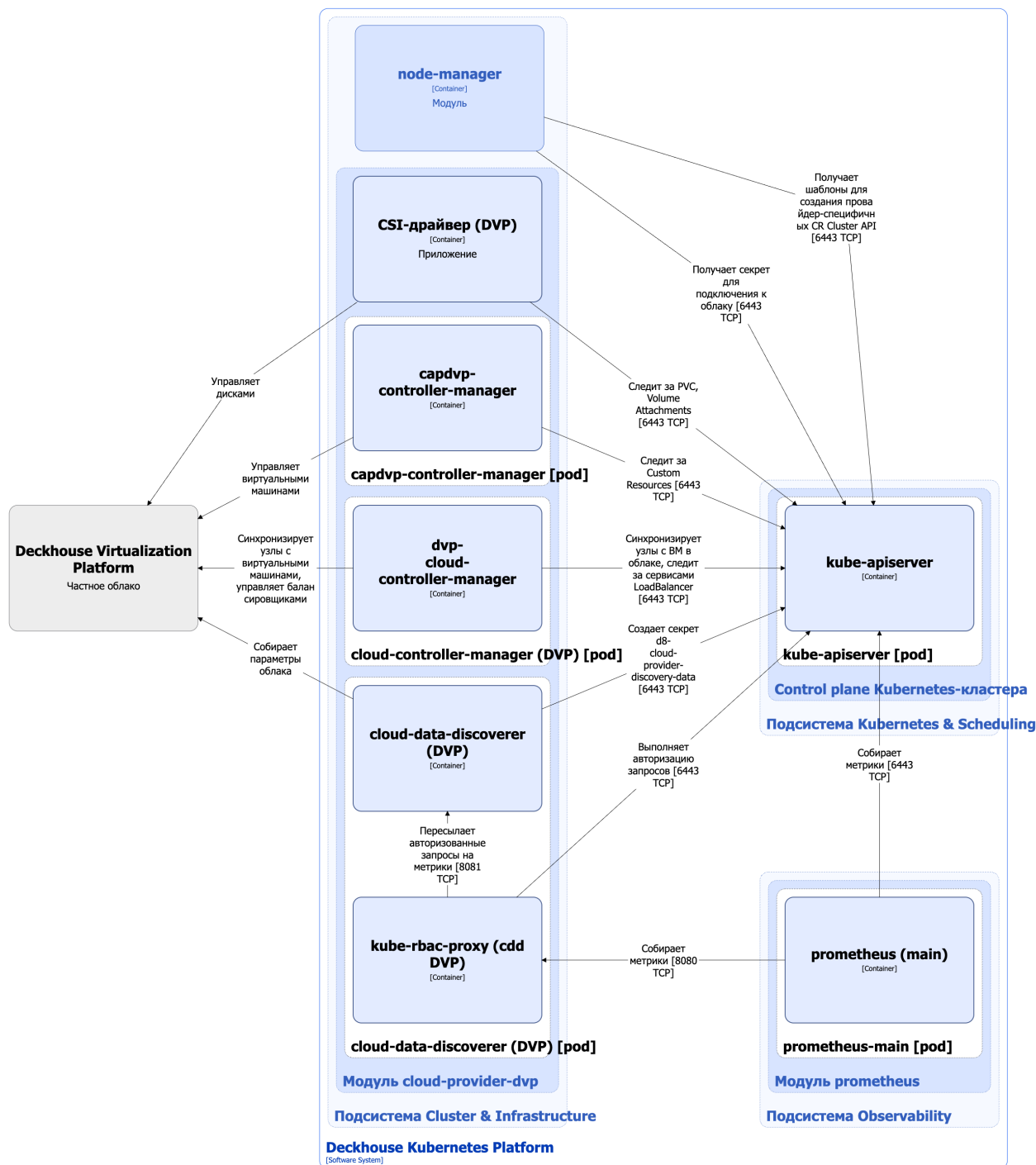
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-dvp` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Capdvp-controller-manager** — Kubernetes Cluster API Provider для DVP. Cluster API является расширением для Kubernetes, которое дает возможность управлять Kubernetes-кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Cluster API Provider позволяет для кластеров под управлением Cluster API заказывать виртуальные машины в инфраструктуре облачного провайдера, в данном случае DVP. Capdvp-controller-manager работает со следующими кастомными ресурсами:
 - DeckhouseCluster — описание кластера на базе DVP;
 - DeckhouseMachineTemplate — шаблон с описанием характеристик создаваемых машин в облаке;
 - DeckhouseMachine — описание характеристик созданной на основе DeckhouseMachineTemplate машины.

Состоит из одного контейнера:

- **capdvp-controller-manager.**

2. **Cloud-controller-manager** — реализация Cloud controller manager для DVP. Обеспечивает

взаимодействие с облаком DVP и выполняет следующие функции:

- реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **dvp-cloud-controller-manager**.

3. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-dvp`. Например, для DVP — это такие параметры, как список зон доступности, ресурсов StorageClass и т.д.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

4. **CSI-драйвер (DVP)** — реализация CSI-драйвера для DVP. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (DVP) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- работа с кастомными ресурсами DeckhouseCluster, DeckhouseMachineTemplate, DeckhouseMachine;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. Облако DVP:

- получение параметров облака;
- управление виртуальными машинами;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

1. Prometheus-main — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-dvp` предоставляет модулю `node-manager` следующие артефакты:

- шаблоны для создания кастомных ресурсов Cluster API для конкретного провайдера, которые `cloud-provider-dvp` использует для создания виртуальных машин в облаке;
- секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах Cluster API, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль `cloud-provider-dvp` предоставляет компоненты Terraform/OpenTofu для DVP, которые используются при сборке исполняемого файла утилиты dhctl в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;

- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый layout включает три модуля:
 - base-infrastructure — базовая инфраструктура (например, создание сетей), может быть пустым;
 - master-node;
 - static-node.

Дополнительные ресурсы

- Документация модуля `cloud-provider-dvp`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-dynamix

Модуль `cloud-provider-dynamix` управляет взаимодействием с облачными ресурсами Basis Dynamix. Он позволяет модулю `node-manager` использовать ресурсы Basis Dynamix при заказе узлов для описанной группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

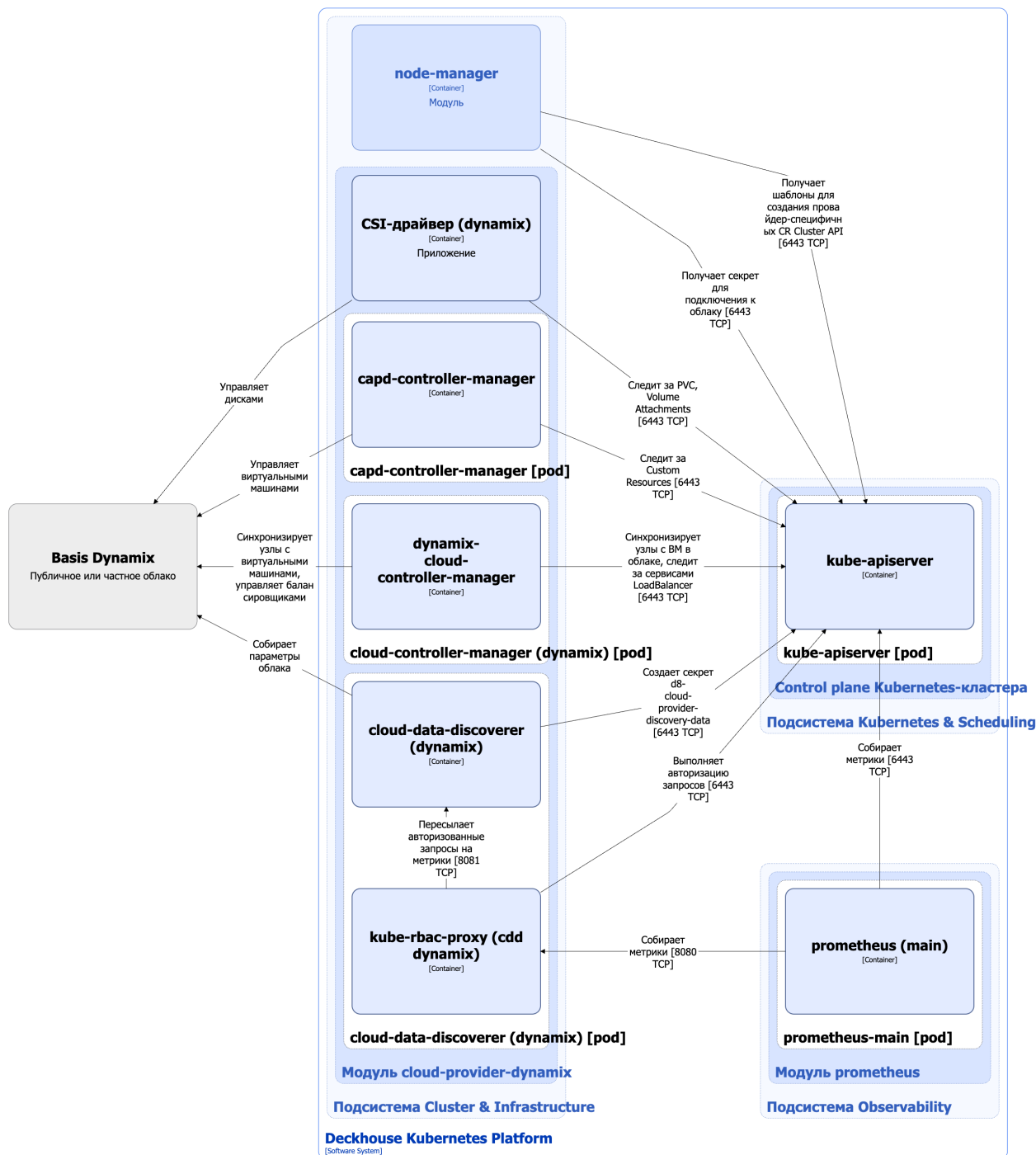
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-dynamix` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Capd-controller-manager** — Kubernetes Cluster API Provider для Basis Dynamix. Cluster API является расширением для Kubernetes, которое дает возможность управлять Kubernetes-кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Cluster API Provider позволяет для кластеров под управлением Cluster API заказывать виртуальные машины в инфраструктуре облачного провайдера, в данном случае Basis Dynamix. Capd-controller-manager работает со следующими кастомными ресурсами:
 - DynamixCluster — описание кластера на базе Basis Dynamix;
 - DynamixMachineTemplate — шаблон с описанием характеристик создаваемых машин в облаке;
 - DynamixMachine — описание характеристик созданной на основе DynamixMachineTemplate машины.

Состоит из одного контейнера:

- **capd-controller-manager**.

2. **Cloud-controller-manager** — реализация Cloud controller manager для Basis Dynamix. Обеспечивает взаимодействие с облаком Basis Dynamix и выполняет следующие функции:

- реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **dynamix-cloud-controller-manager**.

3. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-dynamix`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

4. **CSI-драйвер (dynamix)** — реализация CSI-драйвера для Basis Dynamix. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (dynamix) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- работа с кастомными ресурсами DynamixCluster, DynamixMachineTemplate, DynamixMachine;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. Облако Basis Dynamix:

- получение параметров облака;
- управление виртуальными машинами;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-dynamix` предоставляет модулю `node-manager` следующие артефакты:

- шаблоны для создания кастомных ресурсов Cluster API для конкретного провайдера, которые `cloud-provider-dynamix` использует для создания виртуальных машин в облаке;
- секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах Cluster API, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль `cloud-provider-dynamix` предоставляет компоненты Terraform/OpenTofu для Basis Dynamix, которые используются при сборке исполняемого файла утилиты dhctl в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного

размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый layout включает три модуля:

- `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
- `master-node`;
- `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-dynamix`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-gcp

Модуль `cloud-provider-gcp` обеспечивает интеграцию с облачными ресурсами Google Cloud Platform. Он используется модулем `node-manager` для заказа узлов в соответствии с настройками группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

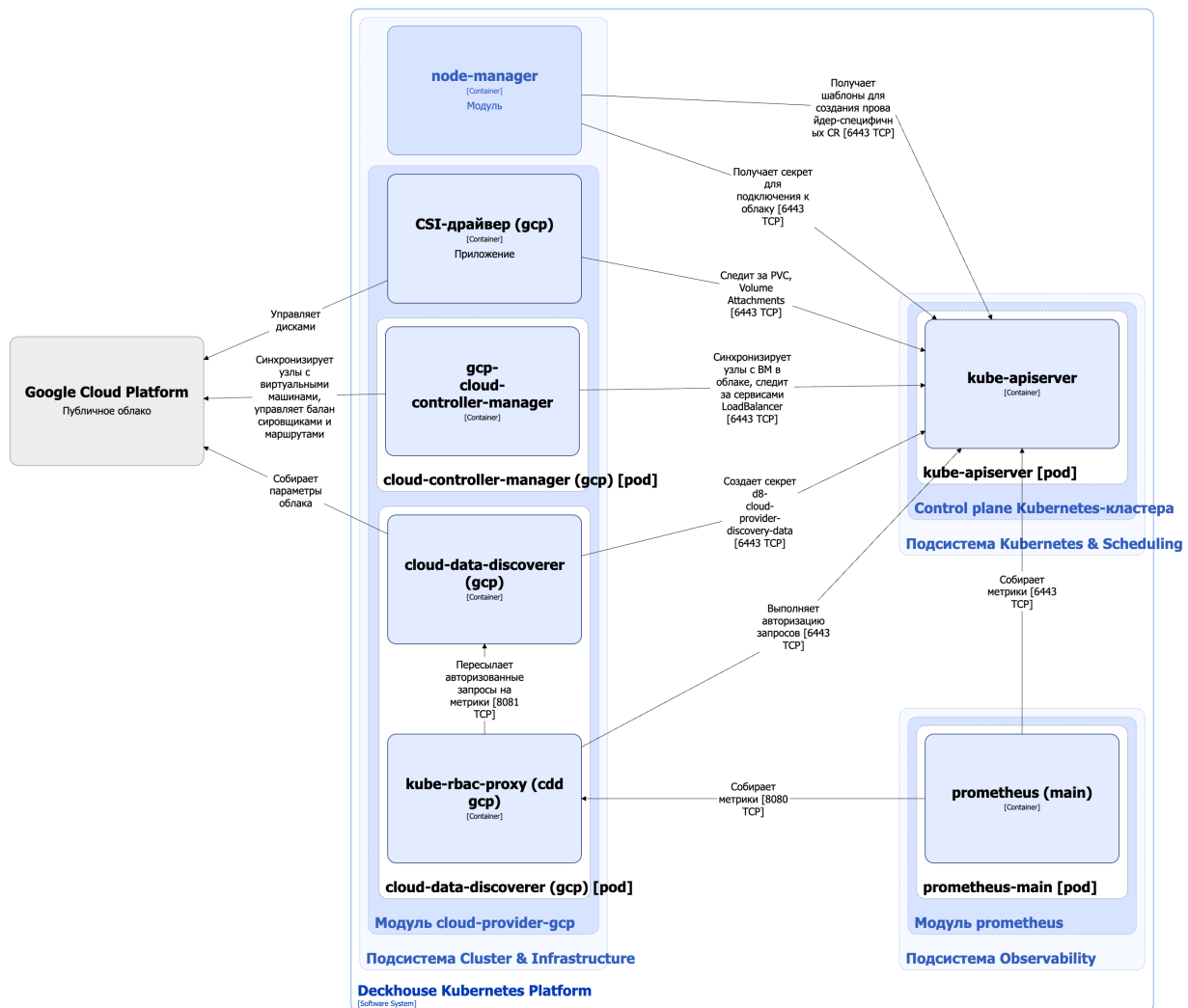
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-gcp` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — реализация cloud controller manager для GCP. Компонент обеспечивает интеграцию с облаком GCP и выполняет следующие функции:
 - реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resurca` Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
 - при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера;
 - создает сетевые маршруты для сети PodNetwork на стороне GCP.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **gcp-cloud-controller-manager**.

2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-gcp`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

3. **CSI-драйвер (gcp)** — реализация CSI-драйвера для GCP. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться в соответствующем разделе документации.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- создание секрета kube-system/d8-cloud-provider-discovery-data;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. Google Cloud Platform:

- получение параметров облака;
- получение ProviderID и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- создание сетевых маршрутов для сети PodNetwork;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль cloud-provider-gcp предоставляет модулю node-manager следующие артефакты:

- шаблоны для создания кастомных ресурсов для конкретного провайдера, которые cloud-provider-gcp использует для создания виртуальных машин в облаке;
- секрет kube-system/d8-node-manager-cloud-provider, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль cloud-provider-gcp предоставляет компоненты Terraform/OpenTofu для GCP, которые используются при сборке исполняемого файла утилиты dhctl в модуле terraform-manager, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждая схема включает три модуля:
 - base-infrastructure — базовая инфраструктура (например, создание сетей), может быть пустым;
 - master-node;
 - static-node.

Дополнительные ресурсы

- Документация модуля cloud-provider-gcp
- Документация модуля node-manager
- Документация модуля terraform-manager

Модуль cloud-provider-huaweicloud

Модуль cloud-provider-huaweicloud управляет взаимодействием с облачными ресурсами Huawei Cloud. Он позволяет модулю node-manager использовать ресурсы Huawei Cloud при заказе узлов для описанной группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

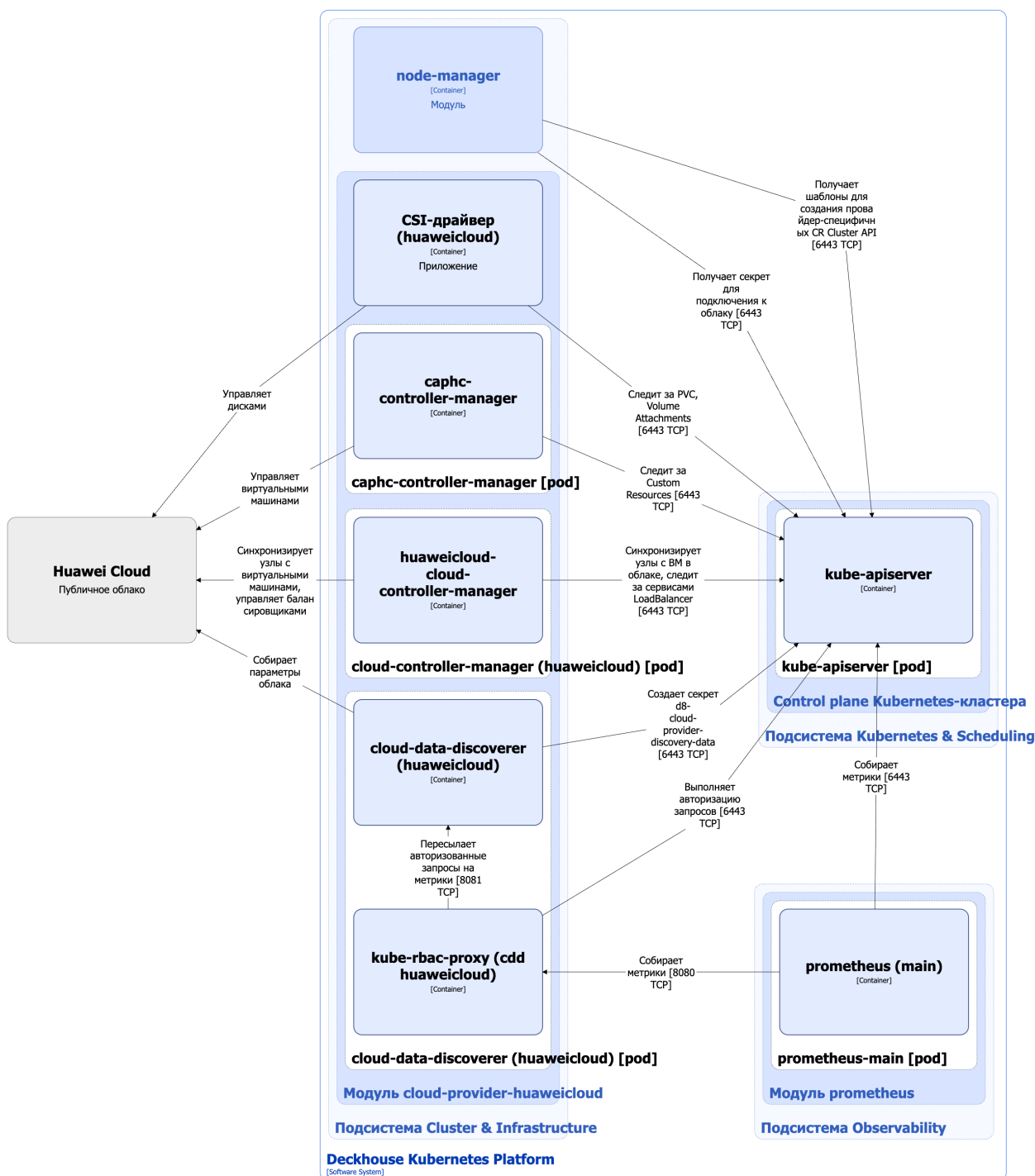
Архитектура модуля



Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-huaweicloud` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Caphc-controller-manager** — Kubernetes Cluster API Provider для Huawei Cloud. Cluster API является расширением для Kubernetes, которое дает возможность управлять Kubernetes-кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Cluster API Provider

позволяет для кластеров под управлением Cluster API заказывать виртуальные машины в инфраструктуре облачного провайдера, в данном случае Huawei Cloud. Caphc-controller-manager работает со следующими кастомными ресурсами:

- HuaweiCloudCluster — описание кластера на базе Huawei Cloud;
- HuaweiCloudMachineTemplate — шаблон с описанием характеристик создаваемых машин в облаке;
- HuaweiCloudMachine — описание характеристик созданной на основе HuaweiCloudMachineTemplate машины.

Состоит из одного контейнера:

- **caphc-controller-manager**.

2. **Cloud-controller-manager** — реализация Cloud controller manager для Huawei Cloud.

Обеспечивает взаимодействие с облаком Huawei Cloud и выполняет следующие функции:

- реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **huaweicloud-cloud-controller-manager**.

3. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-huaweicloud`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

4. **CSI-драйвер (huaweicloud)** — реализация CSI-драйвера для Huawei Cloud. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (huaweicloud) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- работа с кастомными ресурсами HuaweiCloudCluster, HuaweiCloudMachineTemplate, HuaweiCloudMachine;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. **Huawei Cloud:**

- получение параметров облака;
- управление виртуальными машинами;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-huaweicloud` предоставляет модулю `node-manager` следующие артефакты:
 - шаблоны для создания кастомных ресурсов Cluster API для конкретного провайдера, которые `cloud-provider-huaweicloud` использует для создания виртуальных машин в облаке;
 - секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах Cluster API, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.
2. Модуль `cloud-provider-huaweicloud` предоставляет компоненты Terraform/OpenTofu для Huawei Cloud, которые используются при сборке исполняемого файла утилиты dhctl в модуле `terraform-manager`, такие как:
 - Terraform/OpenTofu-провайдер;
 - Terraform-модули;
 - layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый layout включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-huaweicloud`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-openstack

Модуль `cloud-provider-openstack` управляет взаимодействием с облачными ресурсами на базе OpenStack. Он позволяет модулю `node-manager` использовать ресурсы OpenStack при заказе узлов для описанной группы узлов.

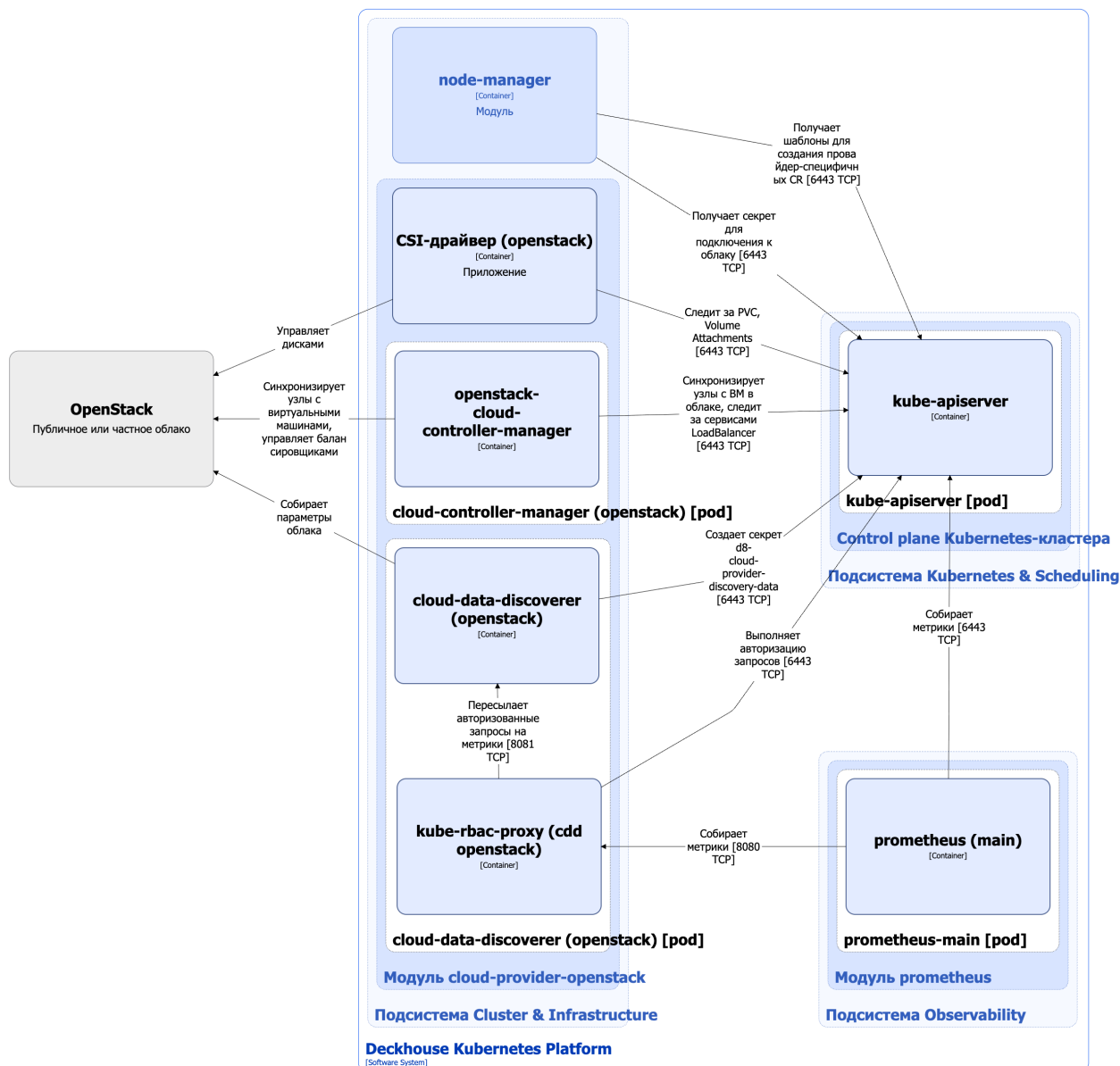
Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-openstack` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — Kubernetes external cloud provider for OpenStack, реализация Cloud controller manager для OpenStack. Обеспечивает взаимодействие с облаком на базе OpenStack и выполняет следующие функции:
 - реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
 - при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера;
 - создает сетевые маршруты для сети PodNetwork на стороне OpenStack.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **openstack-cloud-controller-manager**.

2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-openstack`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.
3. **CSI-драйвер (openstack)** — реализация CSI-драйвера для OpenStack. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (openstack) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг песцпов PersistentVolumeClaim, VolumeAttachment;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. OpenStack:

- получение параметров облака;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- создание сетевых маршрутов для сети `PodNetwork`;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

1. Prometheus-main — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-openstack` предоставляет модулю `node-manager` следующие артефакты:

- шаблоны для создания кастомных ресурсов для конкретного провайдера, которые `cloud-provider-openstack` использует для создания виртуальных машин в облаке;
- секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль `cloud-provider-openstack` предоставляет компоненты Terraform/OpenTofu для OpenStack, которые используются при сборке исполняемого файла утилиты dhctl в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый layout включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-openstack`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-vcd

Модуль `cloud-provider-vcd` обеспечивает интеграцию с облачными ресурсами VMware Cloud Director. Он используется модулем `node-manager` для заказа узлов в соответствии с настройками группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

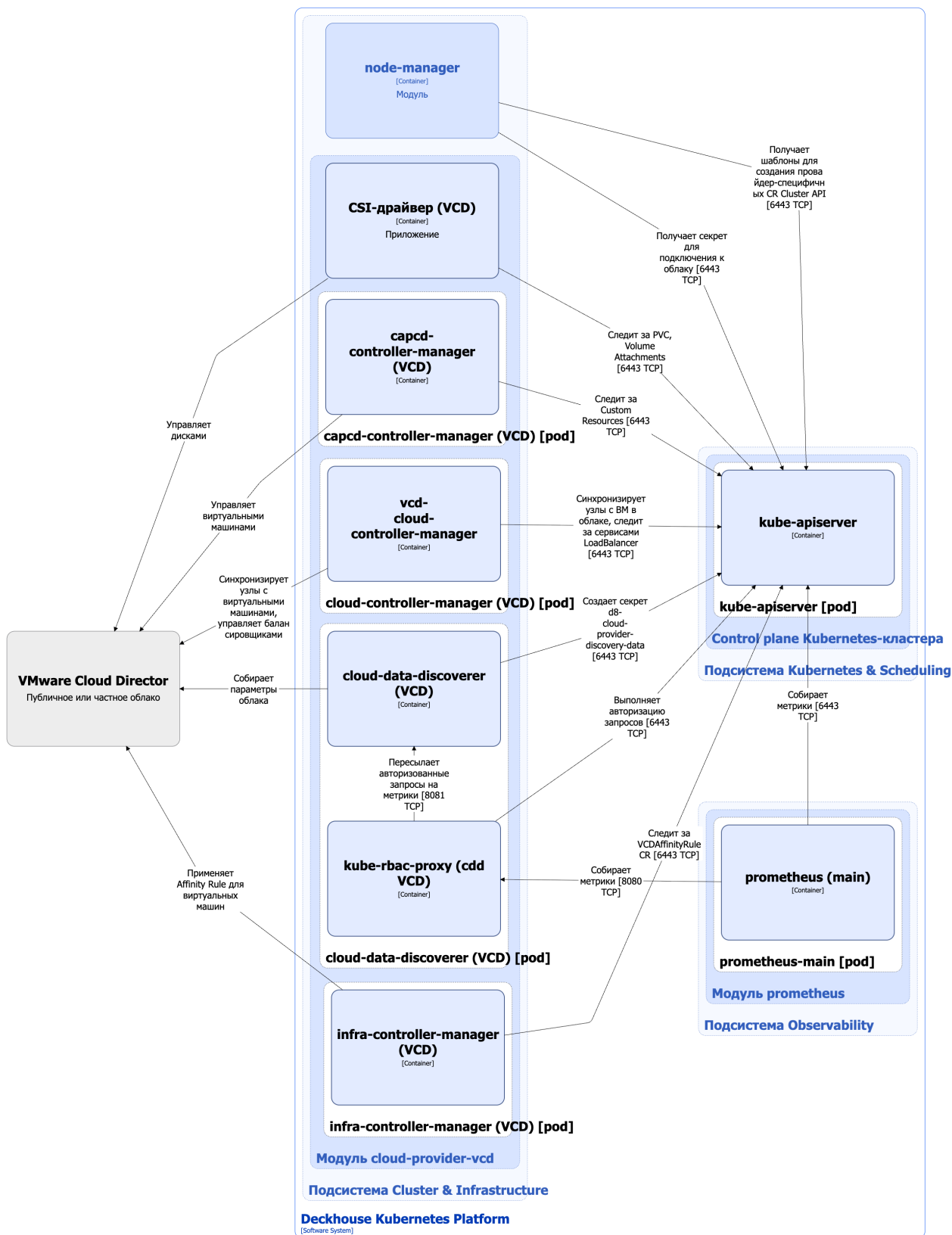
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-vcd` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Capcd-controller-manager** — Kubernetes Cluster API Provider Cloud Director. Cluster API является расширением для Kubernetes, которое дает возможность управлять Kubernetes-кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Cluster API Provider используется для заказа виртуальных машин в инфраструктуре VMware Cloud Director для кластеров, находящихся под управлением Cluster API. Capcd-controller-manager работает со следующими кастомными ресурсами:
 - VCDClusterTemplate — шаблон с описанием инфраструктурных настроек создаваемого кластера (Control Plane endpoint, Loadbalancer, VCD-специфичные настройки);
 - VCDCluster — описание созданного на основе VCDClusterTemplate кластера;

- VCDMachineTemplate — шаблон с описанием характеристик создаваемых машин в облаке;
- VCDMachine — описание характеристик созданной на основе VCDMachineTemplate машины.

Состоит из одного контейнера:

- **capcd-controller-manager**.

2. **Cloud-controller-manager** — Kubernetes External Cloud Provider for VMware Cloud Director, реализация cloud controller manager для VMware Cloud Director. Компонент обеспечивает интеграцию с облаком VMware Cloud Director и выполняет следующие функции:

- реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resourc` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **vcd-cloud-controller-manager**.

3. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-vcd`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

4. **Infra-controller-manager** — отвечает за контроль расположения узлов друг относительно друга на уровне гипервизоров, что может повысить отказоустойчивость и контролировать распределение рабочих нагрузок. Infra-controller-manager работает с кастомным ресурсом VCDAffinityRule, который описывает правила размещения ресурсов в VMware Cloud Director.

Состоит из одного контейнера:

- **infra-controller-manager**.

5. **CSI-драйвер (VCD)** — реализация CSI-драйвера для VMware Cloud Director. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться в соответствующем разделе документации. В модуле `cloud-provider-vcd` используется CSI driver for VMware Cloud Director Named Independent Disks.

CSI-драйвер (VCD) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- работа с кастомными ресурсами VCDClusterTemplate, VCDCluster, VCDMachineTemplate, VCDMachine, VCDAffinityRule;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. **VMware Cloud Director:**

- получение параметров облака;
- управление виртуальными машинами;
- применение правил Affinity Rules к виртуальным машинам;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-vcd` предоставляет модулю `node-manager` следующие артефакты:
 - шаблоны для создания кастомных ресурсов Cluster API для конкретного провайдера, которые `cloud-provider-vcd` использует для создания виртуальных машин в облаке;
 - секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах Cluster API, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.
2. Модуль `cloud-provider-vcd` предоставляет компоненты Terraform/OpenTofu для VMware Cloud Director, которые используются при сборке исполняемого файла утилиты `dhctl` в модуле `terraform-manager`, такие как:
 - Terraform/OpenTofu-провайдер;
 - Terraform-модули;
 - `layouts` — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждая схема включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-vcd`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-vsphere

Модуль `cloud-provider-vsphere` обеспечивает интеграцию с облачными ресурсами VMware vSphere. Он используется модулем `node-manager` для заказа узлов в соответствии с настройками группы узлов.

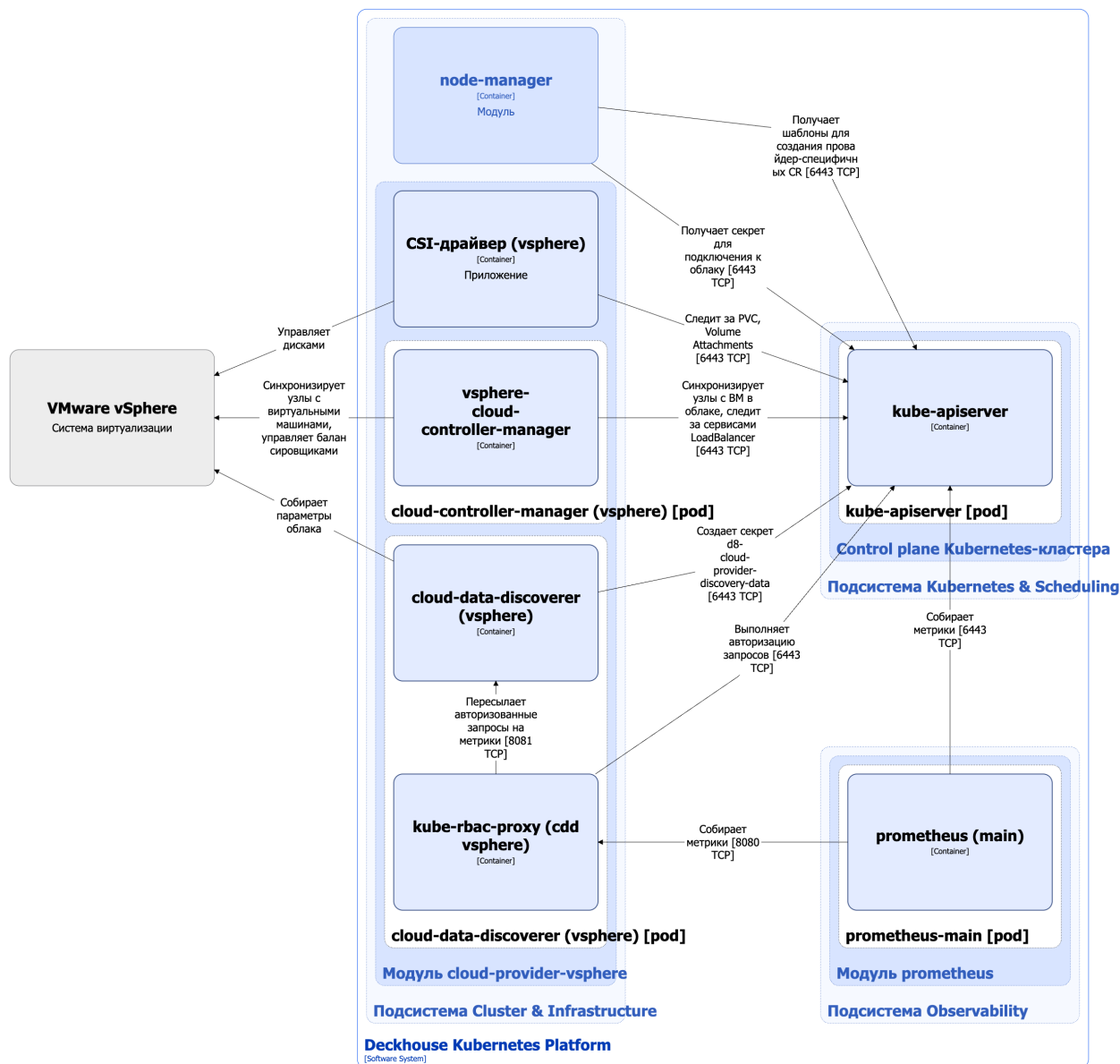
Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-vsphere` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — Kubernetes vSphere Cloud Provider, реализация cloud controller manager для VMware vSphere. Компонент обеспечивает интеграцию с облачными ресурсами на базе VMware vSphere и выполняет следующие функции:
 - реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resurca` Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
 - при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **vsphere-cloud-controller-manager**.

2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-vsphere`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.
3. **CSI-драйвер (vsphere)** — реализация CSI-драйвера для VMware vSphere. С архитектурой CSI-драйвера, используемого в модуле `cloud-provider-vsphere` DKP, можно ознакомиться в соответствующем разделе документации.

CSI-драйвер (vsphere) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. VMware vSphere:

- получение параметров облака;
- получение `ProviderID` и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-vsphere` предоставляет модулю `node-manager` следующие артефакты:

- шаблоны для создания кастомных ресурсов для конкретного провайдера, которые `cloud-provider-vsphere` использует для создания виртуальных машин в облаке;
- секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль `cloud-provider-vsphere` предоставляет компоненты Terraform/OpenTofu для VMware vSphere, которые используются при сборке исполняемого файла утилиты `dhctl` в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждая схема включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-vsphere`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-yandex

Модуль `cloud-provider-yandex` управляет взаимодействием с облачными ресурсами Yandex Cloud. Он позволяет модулю `node-manager` использовать ресурсы Yandex Cloud при заказе узлов для описанной группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

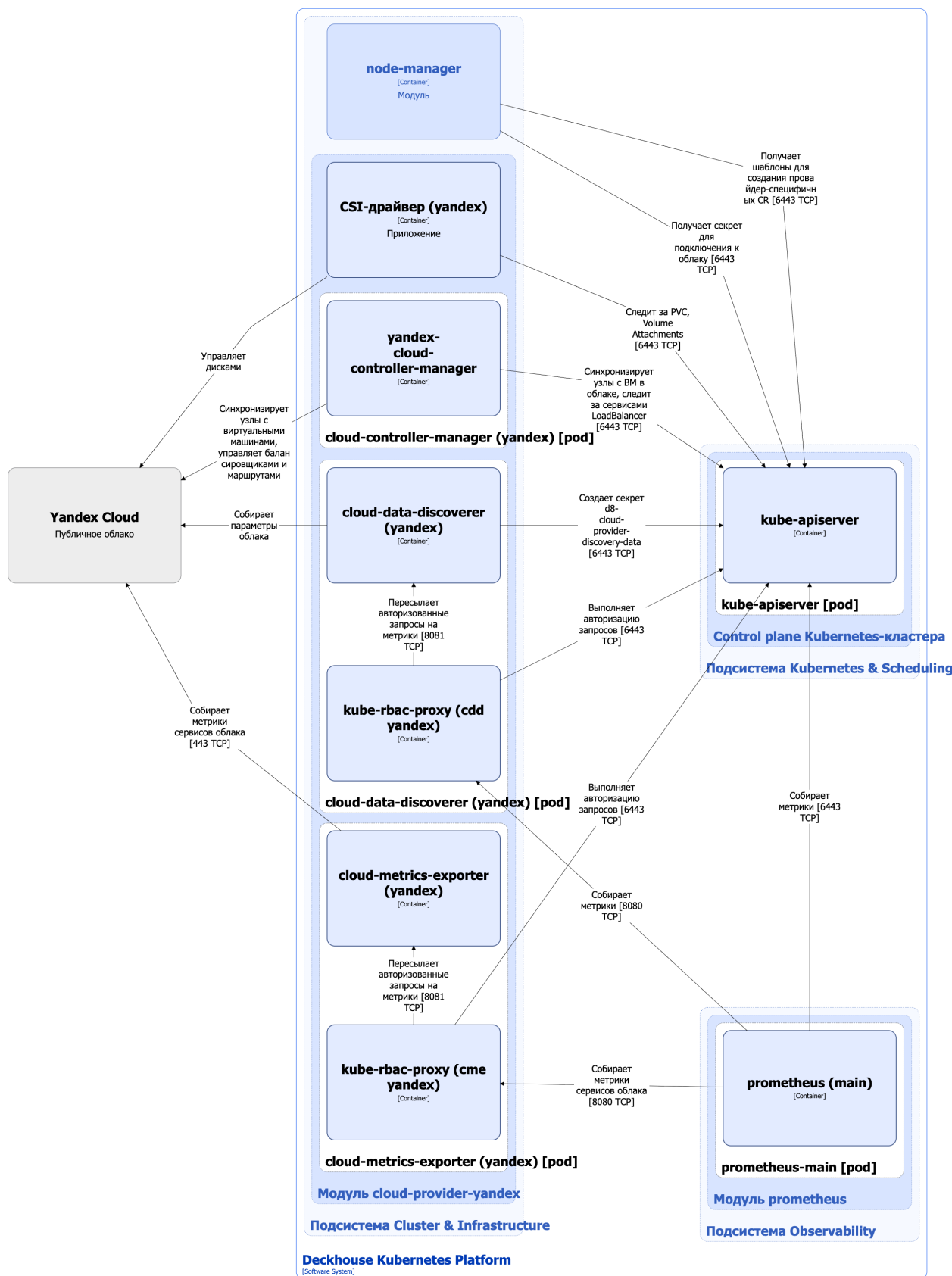
Архитектура модуля



Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-yandex` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-controller-manager** — реализация Cloud controller manager для Yandex Cloud. Обеспечивает взаимодействие с облаком Yandex Cloud и выполняет следующие функции:
 - о реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo.resurca` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс

Node в кластере.

- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера;
- создает сетевые маршруты для сети PodNetwork на стороне Yandex Cloud.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **yandex-cloud-controller-manager**.

2. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета kube-system/d8-cloud-provider-discovery-data. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля cloud-provider-yandex.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.

3. **Cloud-metrics-exporter** — экспортирует метрики сервисов облака Yandex Cloud.

Состоит из следующих контейнеров:

- **cloud-metrics-exporter** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам, получаемым от cloud-metrics-exporter.

4. **CSI-драйвер (yandex)** — реализация CSI-драйвера для Yandex Cloud. С типовой архитектурой CSI-драйвера, используемого в модулях cloud-provider-* DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (yandex) не поддерживает работу со снимками. По этой причине в поде csi-controller отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- создание секрета kube-system/d8-cloud-provider-discovery-data;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. Yandex Cloud:

- получение параметров облака;
- получение ProviderID и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками;
- создание сетевых маршрутов для сети PodNetwork;
- получение метрик сервисов облака.

С модулем взаимодействуют следующие внешние компоненты:

1. Prometheus-main:

- сбор метрик cloud-data-discoverer;
- сбор метрик сервисов облака через cloud-metrics-exporter.

Непрямые взаимодействия:

1. Модуль cloud-provider-yandex предоставляет модулю node-manager следующие артефакты:

- шаблоны для создания кастомных ресурсов для конкретного провайдера, которые cloud-provider-yandex использует для создания виртуальных машин в облаке;
- секрет kube-system/d8-node-manager-cloud-provider, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.

2. Модуль cloud-provider-yandex предоставляет компоненты Terraform/OpenTofu для Yandex Cloud, которые используются при сборке исполняемого файла утилиты dhctl в модуле terraform-manager, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- layouts — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый layout включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-yandex`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль cloud-provider-zvirt

Модуль `cloud-provider-zvirt` управляет взаимодействием с облачными ресурсами zVirt. Он позволяет модулю `node-manager` использовать ресурсы zVirt при заказе узлов для описанной группы узлов.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

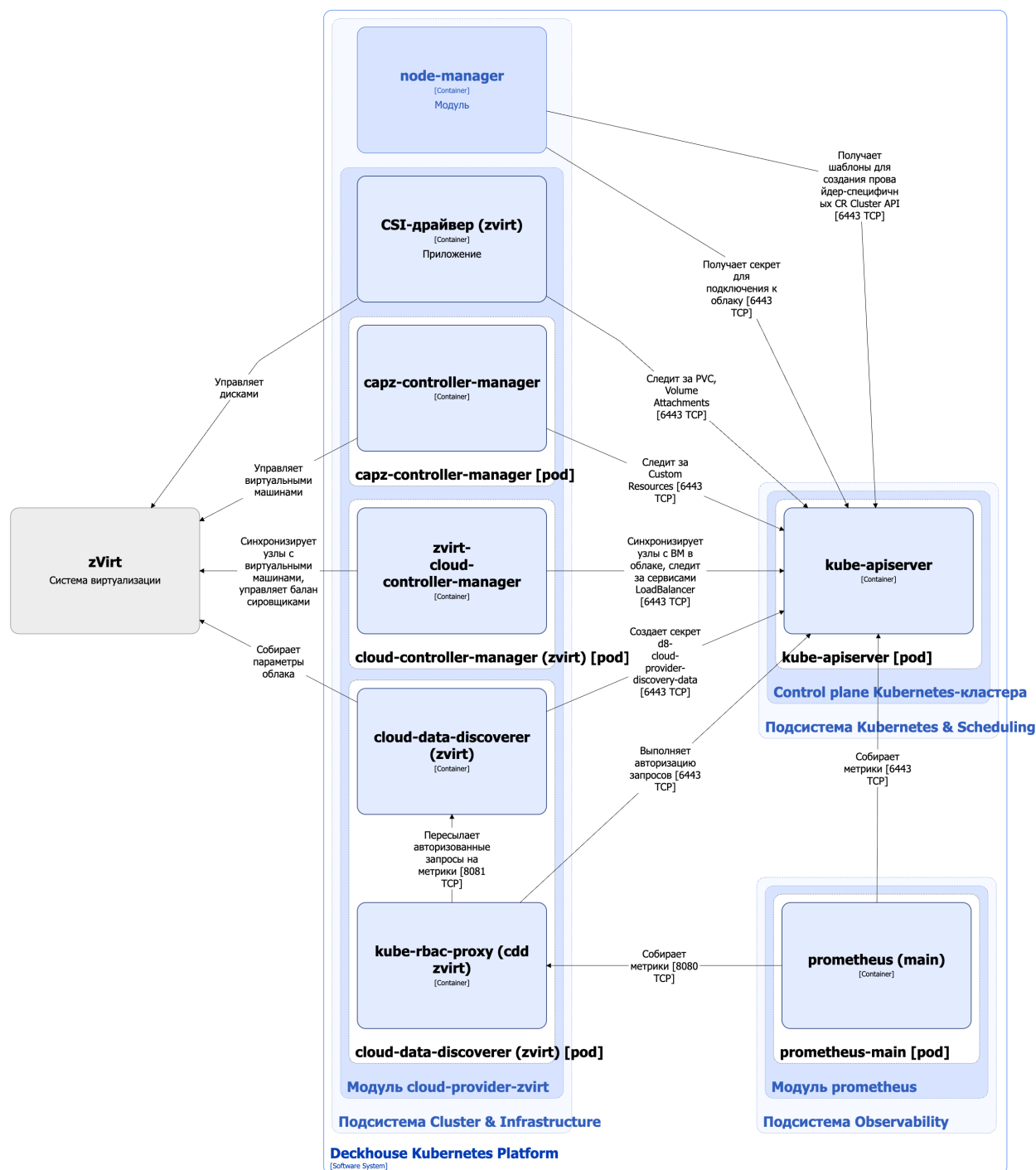
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cloud-provider-zvirt` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Capz-controller-manager** — Kubernetes Cluster API Provider для zVirt. Cluster API является расширением для Kubernetes, которое дает возможность управлять Kubernetes-кластерами как кастомными ресурсами внутри другого Kubernetes-кластера. Cluster API Provider позволяет для кластеров под управлением Cluster API заказывать виртуальные машины в инфраструктуре облачного провайдера, в данном случае zVirt. Capz-controller-manager работает со следующими кастомными ресурсами:
 - ZvirtCluster — описание кластера на базе zVirt;
 - ZvirtMachineTemplate — шаблон с описанием характеристик создаваемых машин в облаке;
 - ZvirtMachine — описание характеристик созданной на основе ZvirtMachineTemplate машины.

Состоит из одного контейнера:

- **capz-controller-manager.**

2. **Cloud-controller-manager** — реализация Cloud controller manager для zVirt. Обеспечивает

взаимодействие с облаком zVirt и выполняет следующие функции:

- реализует связь 1:1 между объектом узла в Kubernetes (Node) и виртуальной машиной в облачном провайдере. Для этого:
 - заполняет поля `spec.providerID` и `nodeInfo` ресурса Node;
 - проверяет наличие виртуальной машины в облаке и при ее отсутствии удаляет ресурс Node в кластере.
- при создании ресурса Service типа LoadBalancer в Kubernetes создаёт балансировщик в облаке, который направляет трафик извне к узлам кластера.

Подробнее о cloud-controller-manager можно почитать в документации Kubernetes.

Состоит из одного контейнера:

- **zvirt-cloud-controller-manager**.

3. **Cloud-data-discoverer** — отвечает за сбор данных из API облачного провайдера и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются другими компонентами модуля `cloud-provider-zvirt`.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cloud-data-discoverer.
4. **CSI-драйвер (zvirt)** — реализация CSI-драйвера для zVirt. С типовой архитектурой CSI-драйвера, используемого в модулях `cloud-provider-*` DKP, можно ознакомиться на соответствующей странице документации.

CSI-драйвер (zvirt) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер snapshotter (external-snapshotter).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;
- работа с кастомными ресурсами ZvirtCluster, ZvirtMachineTemplate, ZvirtMachine;
- создание секрета `kube-system/d8-cloud-provider-discovery-data`;
- синхронизация узлов Kubernetes с виртуальными машинами в облаке;
- мониторинг сервисов типа LoadBalancer;
- авторизация запросов на получение метрик.

2. zVirt:

- получение параметров облака;
- управление виртуальными машинами;
- получение ProviderID и прочей информации о виртуальных машинах, которые являются узлами кластера;
- управление балансировщиками;
- управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик cloud-data-discoverer.

Непрямые взаимодействия:

1. Модуль `cloud-provider-zvirt` предоставляет модулю `node-manager` следующие артефакты:
 - шаблоны для создания кастомных ресурсов Cluster API для конкретного провайдера, которые `cloud-provider-zvirt` использует для создания виртуальных машин в облаке;
 - секрет `kube-system/d8-node-manager-cloud-provider`, в котором содержатся все необходимые настройки для подключения к облаку и создания CloudEphemeral-узлов. Эти настройки прописываются в кастомных ресурсах Cluster API, созданных на основе упомянутых выше шаблонов и учитывающих особенности провайдера.
2. Модуль `cloud-provider-zvirt` предоставляет компоненты Terraform/OpenTofu для zVirt,

которые используются при сборке исполняемого файла утилиты `dhctl` в модуле `terraform-manager`, такие как:

- Terraform/OpenTofu-провайдер;
- Terraform-модули;
- `layouts` — набор схем размещения в облаке, определяющих, как создается базовая инфраструктура, как и с какими дополнительными характеристиками для данного размещения должны создаваться узлы. Например, в одной схеме узлы могут иметь публичные IP-адреса, а в другой — нет. Каждый `layout` включает три модуля:
 - `base-infrastructure` — базовая инфраструктура (например, создание сетей), может быть пустым;
 - `master-node`;
 - `static-node`.

Дополнительные ресурсы

- Документация модуля `cloud-provider-zvirt`
- Документация модуля `node-manager`
- Документация модуля `terraform-manager`

Модуль chrony

Модуль `chrony` обеспечивает синхронизацию времени на всех узлах кластера Deckhouse Kubernetes Platform (DKP) с использованием реализации NTP-сервера/клиента `chrony`.

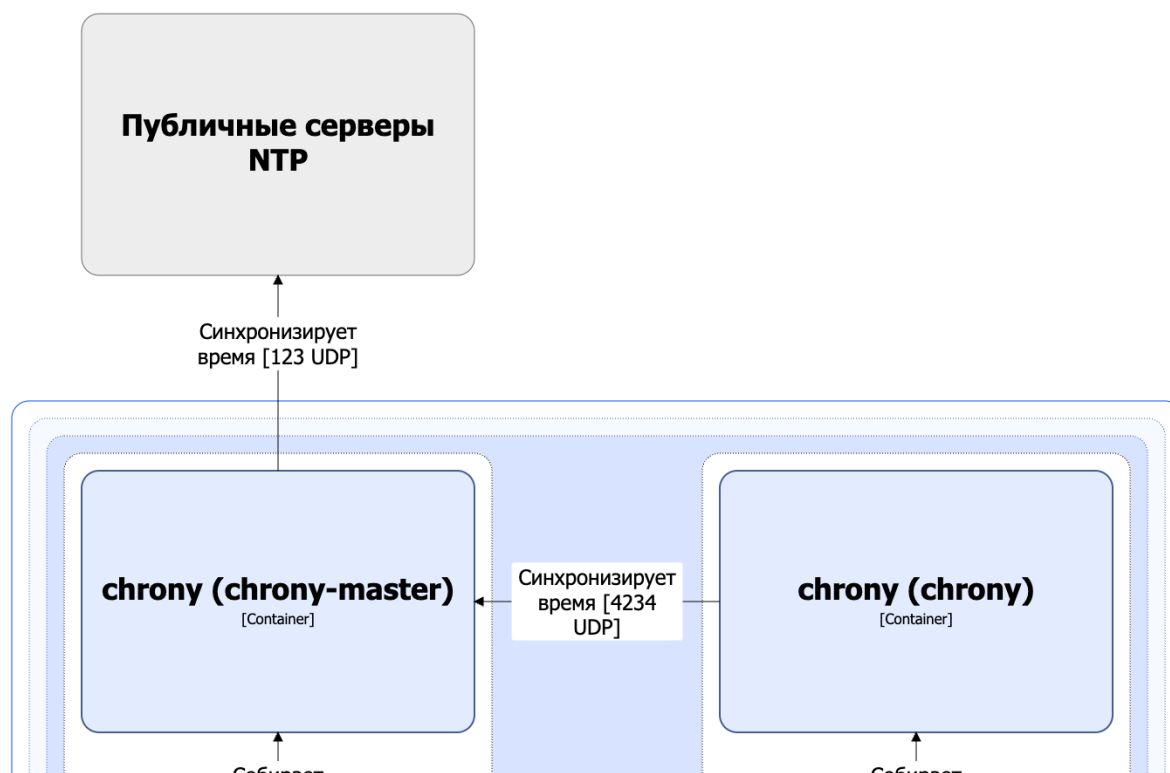
Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

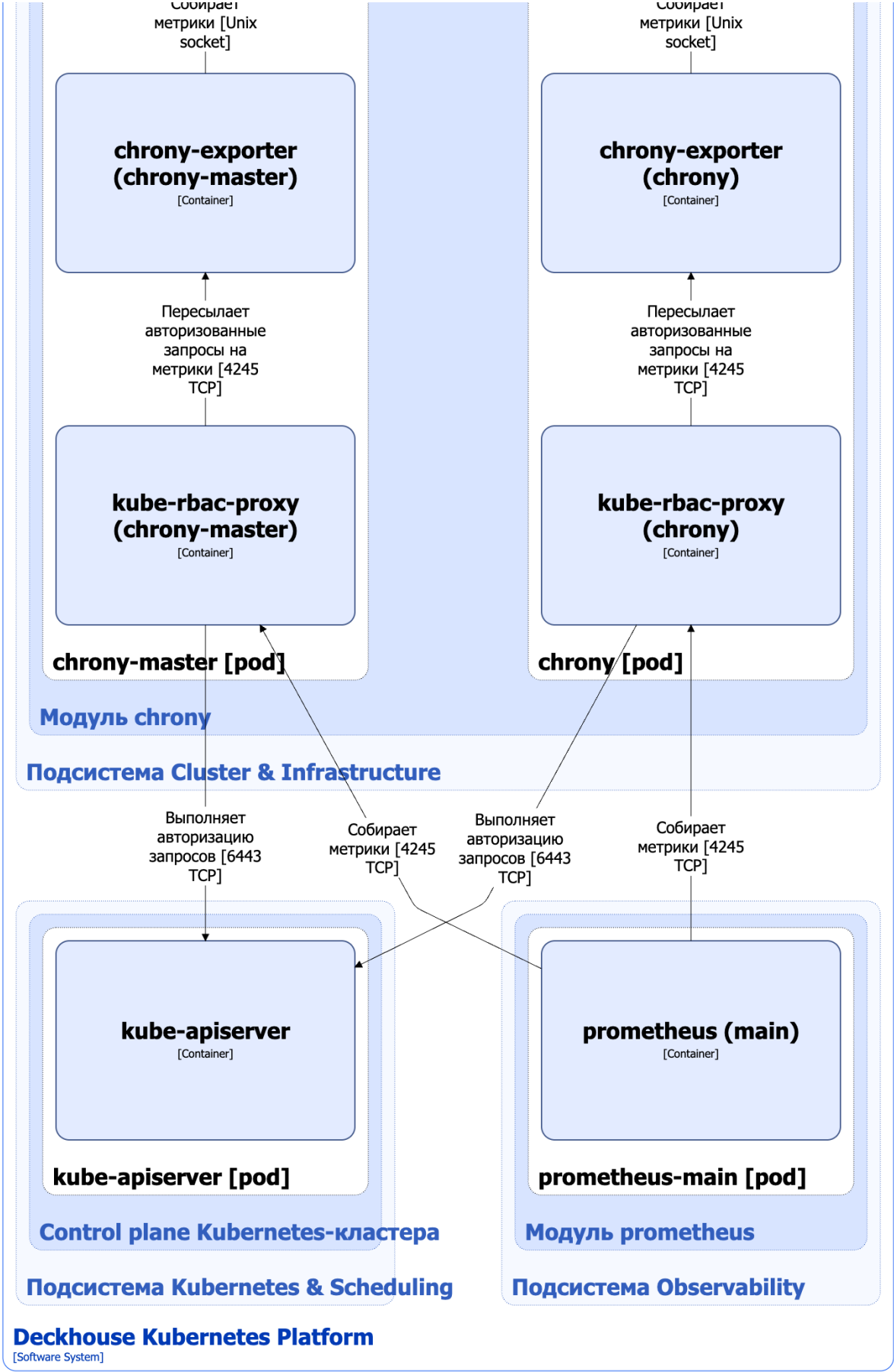
Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `chrony` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:





Компоненты модуля

Модуль состоит из следующих компонентов:

- 1. **Chrony-master** — реализация сервиса синхронизации времени на master-узлах кластера с внешними NTP-серверами.

Состоит из следующих контейнеров:

- **chrony** — основной контейнер;
 - **chrony-exporter** — сайдкар-контейнер, собирающий метрики контейнера chrony и предоставляющий их в формате Prometheus;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера chrony-exporter.
2. **Chrony** — реализация сервиса синхронизации времени на всех узлах кластера, кроме master-узлов, с компонентом chrony-master.

Состоит из следующих контейнеров:

- **chrony** — основной контейнер;
- **chrony-exporter** — сайдкар-контейнер, собирающий метрики контейнера chrony и предоставляющий их в формате Prometheus;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера chrony-exporter.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver** — авторизация запросов на получение метрик.
2. **Внешние NTP-серверы** — синхронизация времени.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик chrony и chrony-master.

Дополнительные ресурсы

- Документация модуля `chrony`

Модуль gpu

Модуль `gpu` обеспечивает управление графическими процессорами (Graphics Processing Units, GPU) в Deckhouse Kubernetes Platform (DKP).

Модуль работает в двух взаимоисключающих режимах, которые переключаются параметром `dra.enabled`:

- режим Dynamic Resource Allocation (DRA) — механизм Kubernetes для запроса и совместного использования устройств, который обеспечивает динамическое и декларативное выделение вычислительных ресурсов GPU;
- режим Device Plugin (по умолчанию) — классическая модель работы с вычислительными ресурсами узла в Kubernetes. В этом режиме модуль публикует ресурсы `nvidia.com/gpu` или `nvidia.com/mig-*`, которые использует kube-scheduler для планирования размещения подов, использующих эти ресурсы.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля зависит от режима работы.

Режим DRA

В режиме DRA модуль построен вокруг не зависящего от поставщика оборудования (вендора) ядра и адаптеров для вендоров NVIDIA и MetaX.

Модуль работает со следующими ресурсами:

- **DeviceClass** — DRA-ресурс, хранящий описание классов устройств, которые могут быть использованы для динамического назначения ресурсов;
- **GPUClass** — кастомный ресурс, который хранит требования к группе GPU (объём памяти, аппаратные возможности) и политику их использования и совместимости;
- **PhysicalGPU** — кастомный ресурс, который хранит описание физического GPU, включая характеристики устройства;
- **ResourceClaim** — DRA-ресурс, который содержит запрос на выделение ресурса для пода и описывает требуемые характеристики и параметры использования;
- **ResourceSlice** — DRA-ресурс, представляющий выделенную долю или часть ресурса, которая назначается в рамках ResourceClaim.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `gpu` в режиме DRA на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **gpu-controller** (Deployment) — контроллер, который реализует обработку запросов на GPU-ресурсы и реализует admission-вебхуки для DRA-объектов через механизм Validating/Mutating Admission Controllers. Контроллер работает на master-узлах.

Контроллер `gpu-controller` выполняет следующие действия:

- создаёт и обновляет DRA-ресурсы DeviceClass на основе кастомных ресурсов PhysicalGPU и GPUClass;
- выполняет валидацию и мутацию запросов на создание и обновление ресурсов Pod, создавая при необходимости ресурсы ResourceClaim;
- отслеживает изменения ресурсов ResourceClaim и на основе этих данных поддерживает состояние и занятость ресурсов PhysicalGPU;
- валидирует ресурсы Pod, GPUClass, ResourceClaim и DeviceClass;
- управляет состоянием PhysicalGPU.

Состоит из следующих контейнеров:

- **gpu-controller** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к основному контейнеру. Kube-rbac-proxy является Open Source-проектом.
2. **gpu-node-agent** (DaemonSet) — компонент, состоящий из одного контейнера **gpu-node-agent**, который выполняет следующие действия:
 - сканирует файловую систему `/sys` хоста и базу идентификаторов PCI;
 - сопоставляет устройства с ConfigMap `gpu-supported-vendors`;
 - создаёт кастомные ресурсы PhysicalGPU для каждого обнаруженного графического процессора;
 - устанавливает лейблы `gpu.deckhouse.io/vendor=<VENDOR>` на ресурсе Node.

Компонент работает на всех узлах кластера, исключая узлы control plane.

3. **<VENDOR>-adapter** (DaemonSet) — компонент, обеспечивающий работу с оборудованием по подготовке, выделению и освобождению ресурсов GPU. На данный момент поддерживаются два вендора: NVIDIA и MetaX.

Компонент выполняет следующие действия:

- регистрируется в kubelet как DRA-плагин kubelet;
- подготавливает и освобождает выделенные ресурсы для подов через операции PrepareResourceClaims и UnprepareResourceClaims;
- публикует список доступных устройств через ресурсы ResourceSlice;
- получает аппаратные возможности оборудования;
- разделяет ресурсы GPU и предоставляет их подам;
- обогащает информацией статус в ресурсах PhysicalGPU.

Состоит из следующих контейнеров:

- **dra-plugin** — сайдкар-контейнер, реализующий DRA-плагин kubelet и обеспечивающий взаимодействие с адаптером вендора;
- **<VENDOR>-adapter** — сайдкар-контейнер, зависящий от вендора и выполняющий взаимодействие с оборудованием на узле;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes

RBAC для организации защищённого доступа к контейнеру `dra-plugin`.

Компонент запускается на всех узлах кластера, у которых есть лейбл `gpu.deckhouse.io/vendor=<VENDOR>`.

4. **gpu-dcgm** (DaemonSet) — компонент, состоящий из одного контейнера **dcgm**, который запускает Data Center GPU Manager (DCGM). DCGM собирает данные о состоянии и использовании GPU (сведения об ошибках (Error Correction Code, ECC), мощность, утилизация). Работает только с графическими процессорами NVIDIA.
5. **gpu-dcgm-exporter** (DaemonSet) — компонент, состоящий из одного контейнера **dcgm-exporter**, который получает метрики GPU из компонента `gpu-dcgm` и отдаёт их в Prometheus-формате.
6. **vfio-switch-<NODE_NAME>-<PCI>** (Job) — компонент, состоящий из одного контейнера **switch**, который переключает используемый драйвер с `nvidia` на `vfio-pci` и наоборот. Создаётся компонентом `nvidia-adapter` для контроля процесса переключения.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - авторизация запросов на получение метрик;
 - работа с кастомными ресурсами `PhysicalGPU` и `GPUClass`;
 - обновление ресурсов `Node`;
 - валидация ресурсов `Pod`, `GPUClass`, `ResourceClaim` и `DeviceClass`;
 - работа с ресурсами `DeviceClass`, `ResourceClaim` и `ResourceSlice`;
 - создание и контроль выполнения Job `vfio-switch-<NODE_NAME>-<PCI>`.
2. **Kubelet** — регистрация в kubelet как DRA-плагин kubelet.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kubelet** — вызов gRPC-методов `PrepareResourceClaims` и `UnprepareResourceClaims`.
2. **Kube-apiserver** — валидация ресурсов `Pod`, `GPUClass`, `ResourceClaim` и `DeviceClass`.
3. **Prometheus-main** — сбор метрик с компонентов `gpu-dcgm` и `<VENDOR>-adapter`.

Режим Device Plugin

В режиме Device Plugin модуль состоит из компонентов, работающих только с адаптерами NVIDIA.

Модуль взаимодействует со следующими ресурсами:

- **NodeFeature** — хранит фактическую информацию об аппаратных возможностях конкретного узла;
- **NodeFeatureRule** — хранит набор правил, на основе которого модуль настраивает лейблы, аннотации и taints для узла кластера.

Архитектура модуля

Архитектура модуля `gpu` в режиме Device Plugin на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **node-feature-discovery-master** (Deployment) — компонент, состоящий из одного контейнера **master**, который собирает информацию об аппаратных возможностях узлов из ресурсов `NodeFeature` и публикует их как лейблы `feature.node.kubernetes.io/*` и `nvidia.com/*` соответствующих узлов. Правила назначения лейблов, taints и аннотаций master получает из ресурсов `NodeFeatureRule`.
2. **node-feature-discovery-worker** (DaemonSet) — компонент, состоящий из одного контейнера **worker**, который запускается на каждом GPU-узле, обнаруживает подключённые PCI- и USB-устройства и публикует информацию о них в виде ресурсов `NodeFeature`. В этих же ресурсах публикуется информация, полученная от компонента `gpu-feature-discovery-<NG>`.

3. **node-feature-discovery-gc** (Deployment) — компонент, состоящий из одного контейнера **gc**, который удаляет устаревшие ресурсы NodeFeature в случае удаления узла.
4. **gpu-feature-discovery-<NG>** (DaemonSet) — компонент опрашивает GPU-драйвер через NVIDIA Management Library (NVML) и записывает информацию об аппаратных возможностях GPU в файл `/etc/kubernetes/node-feature-discovery/features.d/gfd`. `node-feature-discovery-worker` публикует информацию в виде ресурсов NodeFeature, из которых `node-feature-discovery-master` обновляет соответствующие лейблы `nvidia.com/*` для узлов кластера.

Компонент создаётся Deckhouse-контроллером модуля `deckhouse` для каждой NodeGroup (NG), в конфигурации которой указан параметр `.spec.gpu`.

Состоит из следующих контейнеров:

- **gpu-feature-discovery-init** — init-контейнер, подготавливающий конфигурацию для основного контейнера `gpu-feature-discovery-ctr`;
 - **gpu-feature-discovery-ctr** — основной контейнер;
 - **gpu-feature-discovery-sidecar** — сайдкар-контейнер, который отслеживает изменения в конфигурации и перезапускает основной контейнер для применения изменений.
5. **nvidia-device-plugin-<NG>** (DaemonSet) — компонент регистрируется в kubelet через Kubernetes Device Plugin API и публикует GPU-ресурсы для kube-scheduler.

Для работы с ресурсами kubelet вызывает gRPC-методы `ListAndWatch`, `Allocate` и `GetPreferredAllocation` у `nvidia-device-plugin-ctr`. После этого компонент обновляет количество доступных ресурсов на узле и отдаёт эту информацию через kubelet.

Состоит из следующих контейнеров:

- **nvidia-device-plugin-init** — init-контейнер, подготавливающий конфигурацию для основного контейнера `nvidia-device-plugin-ctr`;
 - **nvidia-device-plugin-ctr** — основной контейнер;
 - **nvidia-device-plugin-sidecar** — сайдкар-контейнер, который отслеживает изменения в конфигурации и перезапускает основной контейнер для применения изменений.
6. **nvidia-mig-manager** (DaemonSet) — опциональный компонент, который управляет изменением профиля Multi-Instance GPU (MIG) на узлах с графическими процессорами A100 и H100.

Компонент выполняет следующие действия:

- получает желаемый MIG-профиль (лейбл `nvidia.com/mig.config`) и текущее состояние;
- переводит узел в режим обслуживания (выставляет `taint`, `cordon` или `drain`) при необходимости;
- останавливает поды, использующие GPU на узле;
- применяет MIG-профиль;
- инициирует перезагрузку узла при необходимости;
- возвращает узел в работу.

Состоит из следующих контейнеров:

- **nvidia-mig-manager-init** — init-контейнер, подготавливающий исполняемые файлы и библиотеки;
 - **nvidia-mig-manager** — основной контейнер.
7. **nvidia-dcgm** (DaemonSet) — компонент, состоящий из одного контейнера **nvidia-dcgm**, который запускает Data Center GPU Manager (DCGM). DCGM собирает данные о состоянии и использовании GPU (сведения об ошибках (Error Correction Code, ECC), мощность, утилизация).
 8. **nvidia-dcgm-exporter** (DaemonSet) — компонент, состоящий из одного контейнера **exporter**, который получает метрики GPU из компонента `nvidia-dcgm` и отдаёт их в Prometheus-формате.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - авторизация запросов на получение метрик;
 - работа с ресурсами NodeFeature и NodeFeatureRule;
 - отслеживание и обновление ресурсов Node;

- завершение подов, использующих GPU-ресурсы, при изменении MIG-профиля.

2. **Kubelet** — регистрация через Device Plugin API.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kubelet** — вызов gRPC-методов `ListAndWatch`, `Allocate` и `GetPreferredAllocation`.
2. **Prometheus-main** — сбор метрик `nvidia-dcgm`.

Дополнительные ресурсы

- Документация модуля `deckhouse`
- Документация модуля `gpu`
- Кастомные ресурсы модуля `node-manager`

Модуль registry-packages-proxy

Модуль `registry-packages-proxy` предоставляет сервис HTTP-прокси внутри кластера Deckhouse Kubernetes Platform (DKP) для доступа к пакетам из хранилищ образов контейнеров. Он выступает в качестве посредника между компонентами кластера и внешними или внутренними хранилищами образов контейнеров с функциями кеширования для оптимизации использования пропускной способности сети и повышения производительности при загрузке пакетов.

Модуль является критически важным компонентом инфраструктуры, который работает на master-узлах и используется во время бутстрапа кластера, а также в процессе работы кластера для извлечения пакетов из хранилищ образов контейнеров.

Модуль развёртывает высокодоступный прокси-сервис, который:

- работает на master-узлах с включённым `hostNetwork` для обеспечения доступности во время бутстрапа, когда CNI ещё недоступен;
- предоставляет на порту `4282` отдельный HTTP-эндпоинт только для скачивания `rpp-get` (утилиты для работы с основным эндпоинтом модуля) при бутстрапе узлов, при этом запросы на этот порт не проходят через TLS и `kube-rbac-proxy`, в отличие от основного прокси на `4219`;
- прослушивает порт `4219` (HTTPS) на IP-адресе каждого master-узла;
- предоставляет эндпоинт `GET /package` для извлечения пакетов из хранилища образов контейнеров по дайджесту;
- реализует локальное кеширование извлечённых пакетов (до 1 ГБ) для снижения сетевого трафика и улучшения производительности;
- следит за ресурсом `Secret deckhouse-registry` в неймспейсе `d8-system` для получения учётных данных основного хранилища образов;
- следит за кастомными ресурсами `ModuleSource` и `PackageRepository` для получения учётных данных хранилищ образов контейнеров;
- использует RBAC-авторизацию для защиты доступа к прокси и эндпоинтам метрик;
- предоставляет публичный HTTPS API (через Ingress) для исполняемых файлов и плагинов Deckhouse CLI;
- предоставляет внутрикластерный HTTPS API для иконок пакетов (без публикации через Ingress).

Подробнее с работой модуля можно ознакомиться в разделе описания модуля.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `registry-packages-proxy` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль `registry-packages-proxy` состоит из одного компонента **registry-packages-proxy**, включающего следующие контейнеры:

- **registry-packages-proxy** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к эндпоинтам основного контейнера `registry-packages-proxy`. Является Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - чтение и отслеживание кастомных ресурсов `ModuleSource` и `PackageRepository`;
 - отслеживание `Secret deckhouse-registry`;
 - авторизация запросов на доступ к эндпоинтам основного контейнера.
2. **Хранилище образов** — скачивание образов контейнеров.

С модулем взаимодействуют следующие внешние компоненты:

1. **Bashible** — скачивание образа с собранными исполняемыми файлами на узлы при бутстрапе кластера.
2. **Prometheus-main** — сбор метрик контейнера `registry-packages-proxy`.
3. **Console** — получение иконок пакетов.
4. **Deckhouse CLI** — получение версий утилиты `d8` и её плагинов.

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Документация модуля `registry-packages-proxy`

Модуль terraform-manager

Модуль `terraform-manager` предоставляет инструменты для работы с состоянием Terraform в кластере DKP.

Подробнее с настройками модуля можно ознакомиться в соответствующем разделе документации.

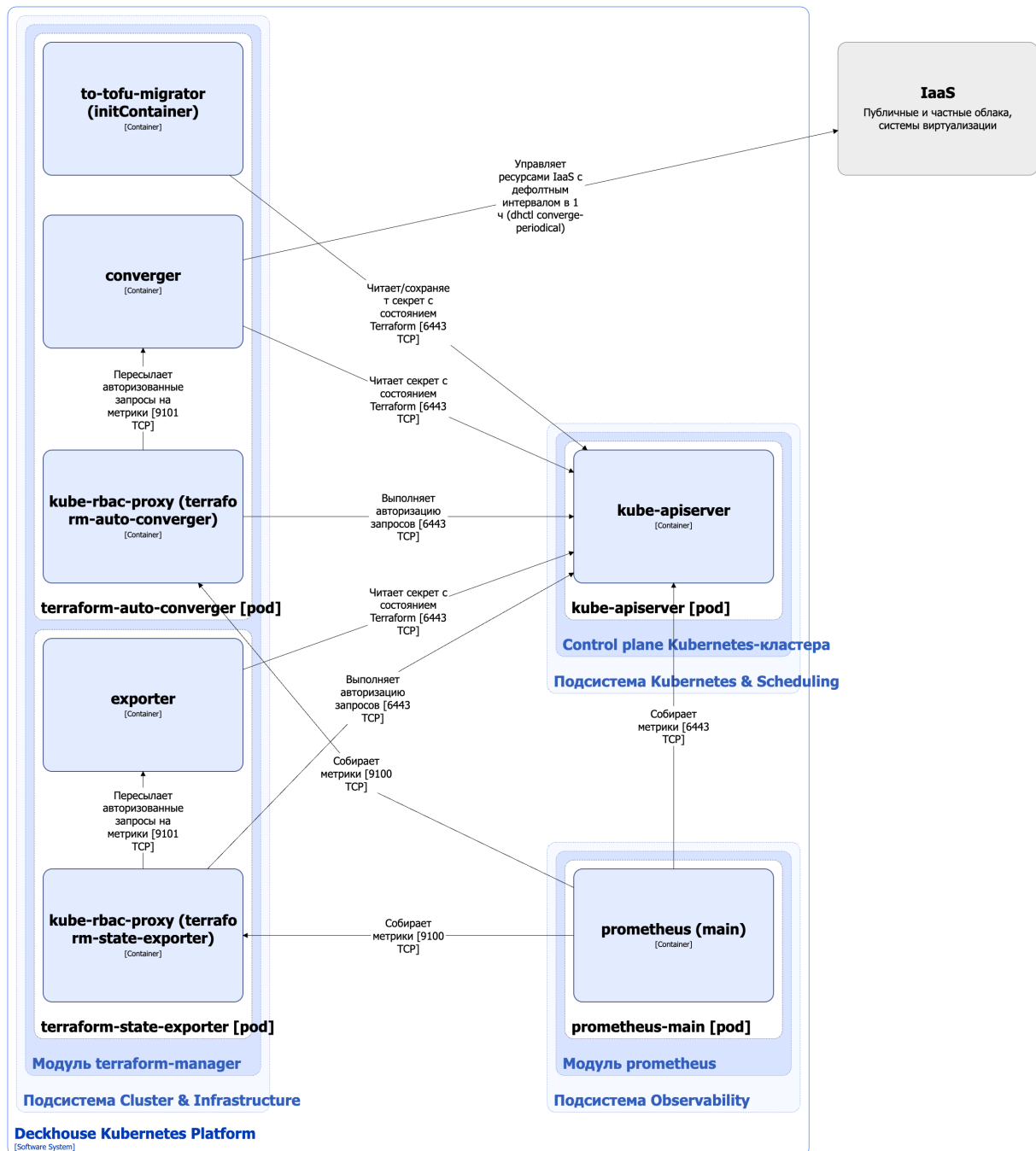
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `terraform-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Terraform-auto-converger** — периодически (по умолчанию раз в час) проверяет состояние Terraform и применяет неdestructивные изменения к ресурсам инфраструктуры.

Компонент работает только с базовой инфраструктурой кластера. Узлы кластера автоматически к требуемому состоянию не приводятся. Периодичность проверки задается параметром `autoConvergerPeriod`.

Состоит из следующих контейнеров:

- **to-tofu-migrator** — init-контейнер для миграции состояния Terraform в OpenTofu. В контейнере запускается утилита `dhctl` с командой `converge-migration`;
- **converger** — основной контейнер, в котором запускается утилита `dhctl` с командой `converge-periodical`;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера `converger`. Является Open Source-проектом.

2. **Terraform-state-exporter** — проверяет состояние Terraform и экспортирует связанные с ним метрики.

Состоит из следующих контейнеров:

- **exporter** — основной контейнер, в котором запускается утилита `dhctl` с командой `terraform converge-exporter`;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера exporter.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- чтение и запись секрета с состоянием Terraform;
- авторизация запросов на получение метрик.

2. Облачная инфраструктура (или система виртуализации) — управляет базовыми инфраструктурными ресурсами и приводит их к желаемому состоянию.

С модулем взаимодействуют следующие внешние компоненты:

- **prometheus-main** — сбор метрик `terraform-auto-converger` и `terraform-state-exporter`.

Дополнительные ресурсы

- Документация модуля `terraform-manager`

Подсистема IAM

В данном подразделе описывается архитектура подсистемы IAM (Identity and Access Management, идентификация и управление доступом) платформы Deckhouse Kubernetes Platform (DKP).

Подсистема IAM отвечает за следующие функции в DKP:

- аутентификация пользователей;
- ролевая модель управления доступом (RBAC);
- мультитенантность;
- автоматическое назначение аннотаций и лейблов неймспейсам.

В подсистему IAM входят следующие модули, реализующие описанные выше функции:

- `user-authn` — аутентификация пользователей;
- `user-authz` — ролевая модель управления доступом;
- `multitenancy-manager` — мультитенантность;
- `namespace-configurator` — автоматическое назначение аннотаций и лейблов неймспейсам.

Аутентификация

Подключение к API Kubernetes с помощью сгенерированного kubeconfig

1. **Инициализация.** До запуска `kube-apiserver` он запрашивает конфигурационный эндпоинт OIDC-провайдера (в данном случае — Dex), чтобы получить информацию об `issuer` и параметры для валидации токенов через JWKS-эндпоинт.
2. **Генерация kubeconfig.** Веб-интерфейс Deckhouse Kubernetes Platform (DKP) формирует `kubeconfig`, в котором сохраняются `ID token` и `refresh token`. Этот файл используется утилитой `kubectl` или другими клиентами Kubernetes.
3. **Аутентификация при обращении к API.** При получении запроса с `ID token`, `kube-apiserver` проверяет его подпись, используя ключи, полученные с JWKS-эндпоинта. Затем проверяются значения `claim'ов iss (issuer)` и `aud (audience)` токена на соответствие конфигурации сервера.

Защита Dex от подбора логина и пароля

Одному пользователю разрешено только 20 попыток входа. Если указанный лимит израсходован, одна дополнительная попытка будет добавляться каждые 6 секунд.

Аутентификация с помощью DexAuthenticator

1. **Процесс входа через Dex.** В большинстве случаев Dex перенаправляет пользователя на страницу входа внешнего провайдера (например, GitHub, Okta, Keycloak) и ожидает, что после успешной аутентификации пользователь будет возвращён на адрес `/callback`. Однако для провайдеров, таких как LDAP или Atlassian Crowd, этот механизм не работает. В таких случаях пользователь вводит логин и пароль в форму Dex, и сам Dex выполняет проверку через API соответствующего провайдера.
2. **Хранение токенов и сессий.** DexAuthenticator устанавливает cookie с полным `refresh token`, а не выдает короткоживущий `ticket`, как это делается с `ID token`. Это связано с тем, что Redis, используемый в DexAuthenticator, не сохраняет данные на диск. Если в Redis не найден `ID token` по `ticket`, пользователь может получить новый `ID token`, предъявив `refresh token`, сохранённый в cookie.
3. **Передача токена в приложение.** DexAuthenticator добавляет HTTP-заголовок `Authorization` с `ID token` из Redis. Это поведение может быть необязательным для некоторых приложений, таких как Upmeter, где механизмы авторизации реализованы иначе. Однако для приложений вроде Kubernetes Dashboard это обязательное поведение, так как Dashboard использует `ID token` для доступа к Kubernetes API от имени пользователя.

Расширения Dex в DKP

DKP использует модифицированную версию Dex для поддержки:

- групп для статических учетных записей пользователей и провайдера Bitbucket Cloud (параметр `bitbucketCloud`);
- передачи параметра `group` клиентам;
- механизма `obsolete tokens`, который позволяет избежать состояния гонки при продлении токена OIDC-клиентом.

Отказоустойчивый режим

DKP поддерживает режим высокой доступности `highAvailability`. При его включении аутентификаторы, отвечающие на `auth request`-запросы, развертываются с учетом требуемой избыточности для обеспечения непрерывной работы. В случае отказа любого из экземпляров аутентификаторов пользовательские аутентификационные сессии не прерываются.

Дополнительные ресурсы

- Кастомные ресурсы модуля `user-authn`

Мультитенантность

Модуль `multitenancy-manager` позволяет создавать изолированные проекты в Deckhouse Kubernetes Platform (DKP). Проекты обеспечивают квоты ресурсов, сетевую изоляцию и функции безопасности, выходящие за рамки стандартных неймспейсов.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Внутренняя логика работы

Создание проекта

Для создания проекта используются следующие кастомные ресурсы:

- `ProjectTemplate` — описывает шаблон проекта. Задается список ресурсов, которые будут созданы в проекте, а также схема параметров, которые можно передать при создании проекта;
- `Project` — описывает конкретный проект.

При создании `Project` из определенного `ProjectTemplate` происходит следующее:

1. Переданные параметры валидируются по OpenAPI-спецификации (параметр `openAPIV3Schema` ресурса `ProjectTemplate`).
2. Выполняется рендеринг шаблона для ресурсов с помощью Helm. Значения для рендеринга берутся из параметра `parameters` ресурса `Project`.
3. Создаётся неймспейс с именем, которое совпадает с именем `Project`.
4. По очереди создаются все ресурсы, описанные в шаблоне.



При изменении шаблона проекта все созданные проекты будут обновлены в соответствии с новым шаблоном.

Изоляция проекта

В основе проекта используется механизм изоляции ресурсов в рамках неймспейса. Неймспейсы позволяют группировать поды, сервисы, секреты и другие объекты, но не обеспечивают полноценной изоляции. Проект расширяет функциональность неймспейсов, предлагая дополнительные инструменты для повышения уровня контроля и безопасности.

Для управления уровнем изоляции проекта можно использовать возможности Kubernetes, например:

- **Ресурсы контроля доступа** (AuthorizationRule / RoleBinding) — позволяют управлять взаимодействием объектов внутри неймспейса. С их помощью можно задавать правила и назначать роли, чтобы точно контролировать, кто и что может делать в проекте.
- **Ресурсы контроля использования нагрузки** (ResourceQuota) — с их помощью можно задать лимиты на использование процессорного времени (CPU), оперативной памяти (RAM), а также количества объектов внутри неймспейса. Это помогает избежать чрезмерной нагрузки и обеспечивает мониторинг за приложениями в рамках проекта.
- **Ресурсы контроля сетевой связности** (NetworkPolicy) — управляют входящим и исходящим сетевым трафиком в неймспейсе. Таким образом, можно настроить разрешенные подключения между подами, улучшить безопасность и управляемость сетевого взаимодействия в рамках проекта.

Эти инструменты можно комбинировать, чтобы настроить проект в соответствии с требованиями вашего приложения.

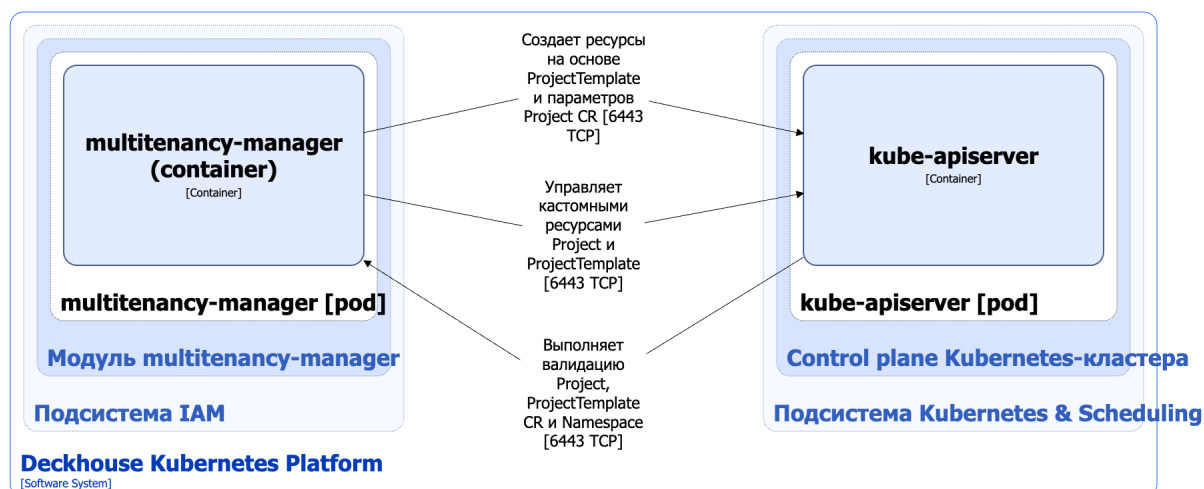
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `multitenancy-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме.



Компоненты модуля

Модуль состоит из следующих компонентов:

- **Multitenancy-manager** — компонент состоит из одного контейнера `multitenancy-`

manager и обеспечивает следующие функции:

- управление кастомными ресурсами Project и ProjectTemplate;
- валидация кастомных ресурсов Project и ProjectTemplate;
- валидация стандартного ресурса Namespace если в параметрах модуля `multitenancy-manager` задано `.spec.settings.allowNamespacesWithoutProjects=false`;
- создание ресурсов, указанных в кастомном ресурсе ProjectTemplate, на основе параметров, заданных в Project.

i

Внимание. Multitenancy-manager имеет права `cluster-admin`, что позволяет создавать любые объекты, описанные в ресурсе ProjectTemplate.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

- **Kube-apiserver:**
 - управление кастомными ресурсами Project и ProjectTemplate;
 - валидация кастомных ресурсов Project, ProjectTemplate, а также стандартного ресурса Namespace;
 - создание ресурсов, указанных в кастомном ресурсе ProjectTemplate, на основе параметров, заданных в Project.

Дополнительные ресурсы

- Документация модуля `multitenancy-manager`

Модуль user-authn

Модуль `user-authn` реализует единую систему аутентификации, интегрированную с Kubernetes и веб-интерфейсами, используемыми в модулях Deckhouse Kubernetes Platform (DKP), например, в модуле `console`.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

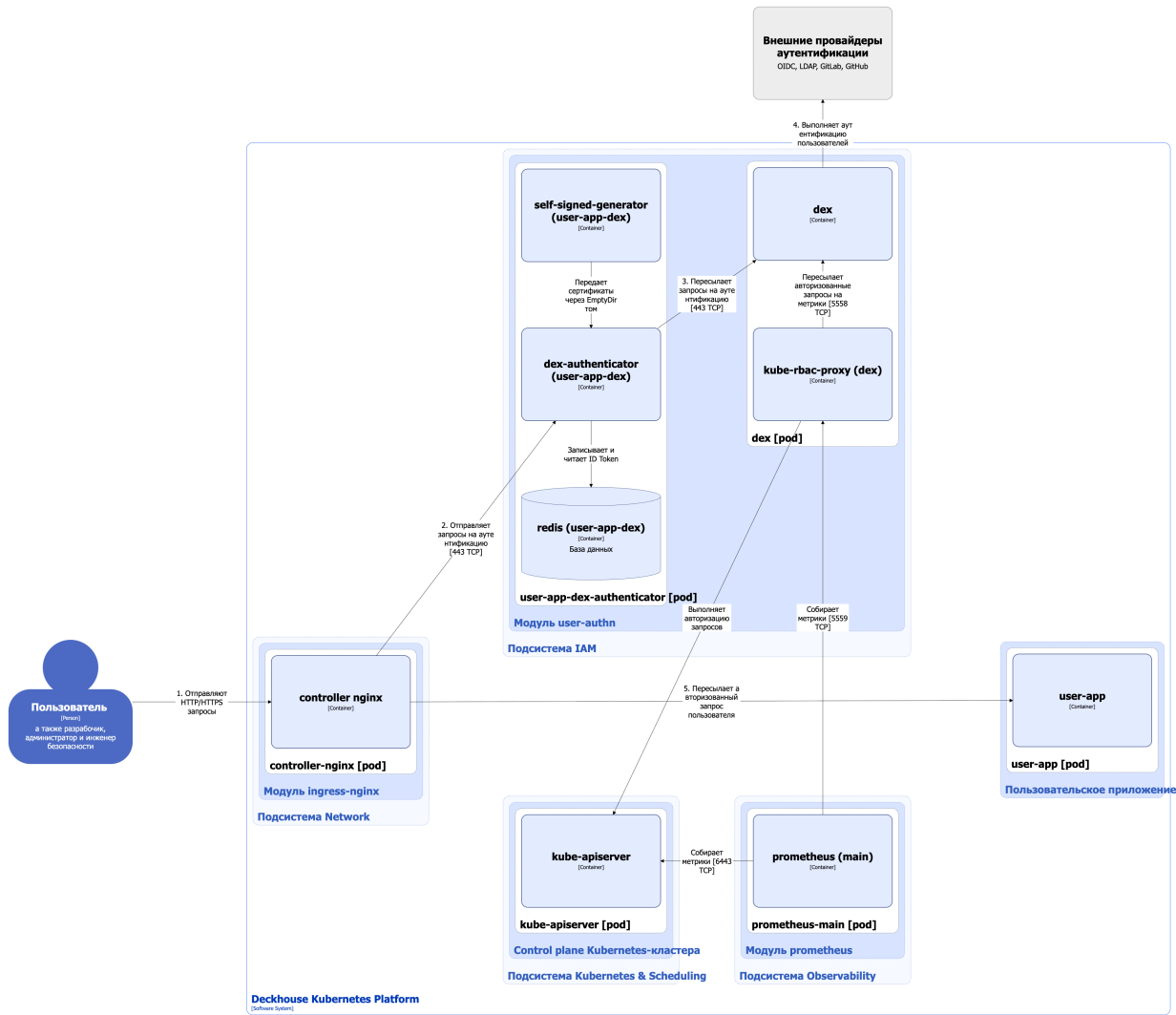
- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

В DKP для служебных сервисов и пользовательских приложений используются две схемы аутентификации:

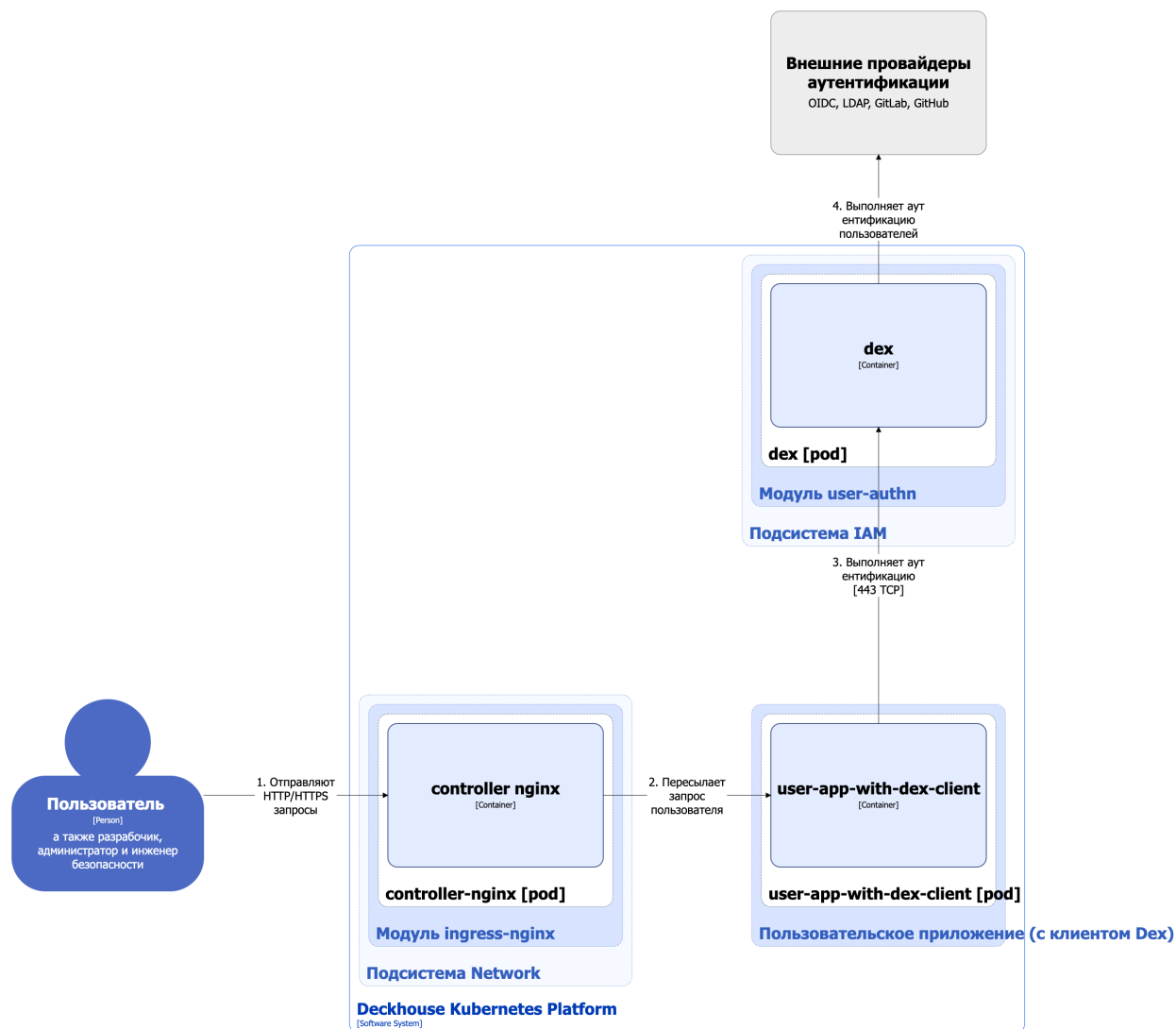
- с использованием `dex-authenticator`;
- с использованием клиента Dex.

Архитектура модуля `user-authn` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующих диаграммах.

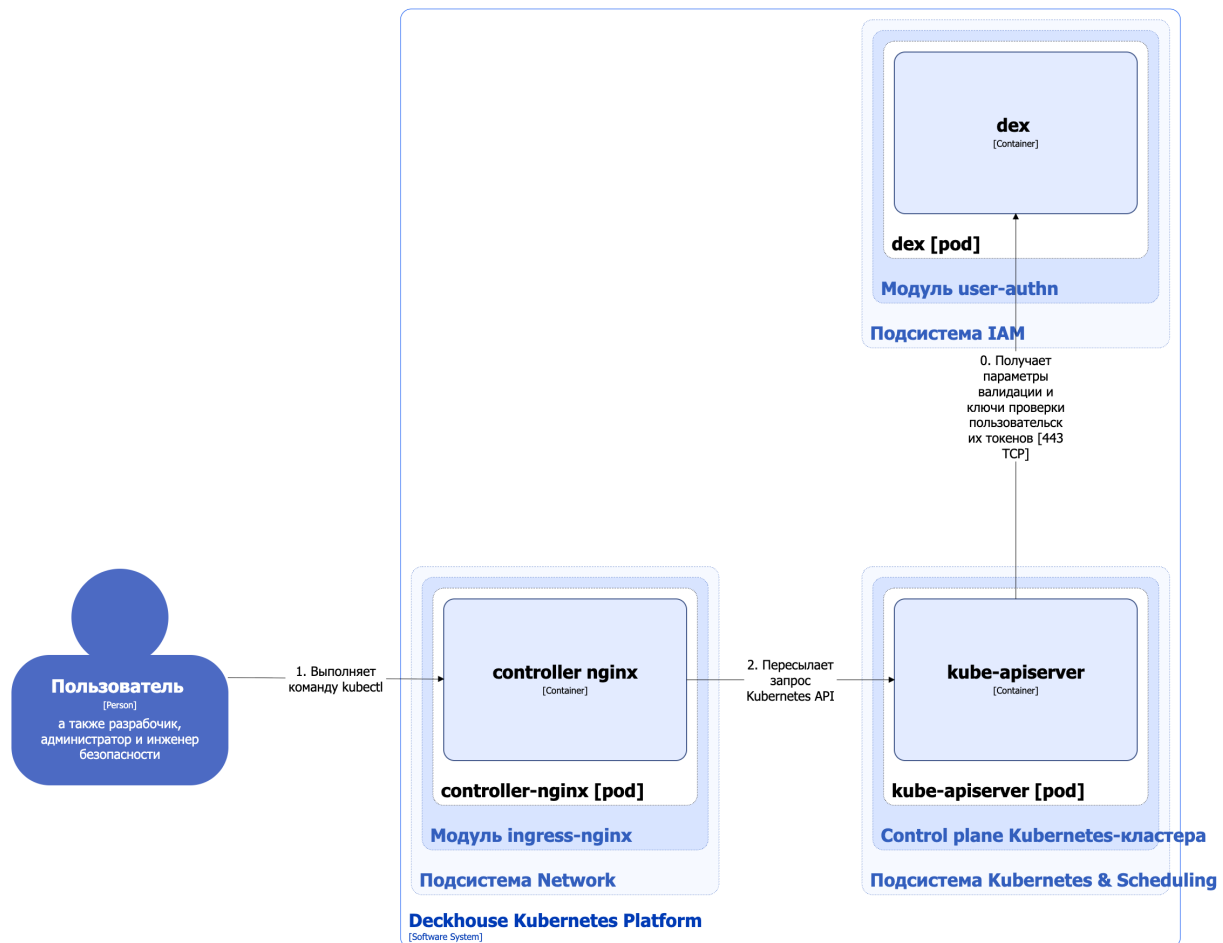
Вариант с использованием `dex-authenticator`:



Вариант с использованием клиента Dex (для упрощения на схеме не показано взаимодействие модуля prometheus-main с dex):



При подключении к API Kubernetes с помощью утилиты `kubectl` или других клиентов Kubernetes с использованием сгенерированного `kubeconfig` используется отдельная схема аутентификации. Она подробно описана в соответствующем разделе документации:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Dex** — федеративный провайдер OpenID Connect, поддерживающий работу со статическими пользователями и интеграцию с внешними провайдерами аутентификации, такими как SAML, GitLab или GitHub.

Состоит из следующих контейнеров:

- **dex** — основной контейнер, реализующий функции Dex;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам провайдера.

2. **Dex-authenticator** — middleware-сервис для аутентификации запросов к приложениям через сервис аутентификации кластера DKP.

При соответствующей настройке Ingress-контроллера (через модуль `auth_request` NGINX) запросы сначала направляются в dex-authenticator для аутентификации.

Состоит из следующих контейнеров:

- **dex-authenticator** — основной контейнер сервиса;
- **redis** — сайдкар-контейнер с базой данных Redis, используемый для временного хранения ID Token и быстрого доступа к ним (за счет размещения базы в памяти);
- **self-signed-generator** — init-контейнер, генерирующий самоподписанный сертификат при запуске пода.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Внешние провайдеры аутентификации.
2. **Kube-apiserver** — авторизация запросов на получение метрик.

С модулем взаимодействуют следующие внешние компоненты:

1. **Ingress-контроллер** — перенаправляет в dex-authenticator запросы на аутентификацию в

служебных сервисах DKP и в пользовательских приложениях.

2. **Пользовательские приложения** — могут аутентифицироваться в dex напрямую (без dex-authenticator), если для приложения настроен OAuth2-клиент в Dex. Подробнее с настройкой клиента Dex можно ознакомиться в документации модуля `user-authn`.
3. **Kube-apiserver** — обращается в dex при обработке запросов к API Kubernetes, выполняемых с использованием файла kubeconfig:
 - при запуске kube-apiserver запрашивает конфигурационный эндпоинт OIDC-провайдера (в данном случае — Dex), чтобы получить информацию об issuer и параметры для проверки токенов через JWKS-эндпоинт;
 - при получении запроса с ID token kube-apiserver проверяет его подпись с использованием ключей, полученных с JWKS-эндпоинта.

Подробнее о том, как работает подключение к API Kubernetes с помощью сгенерированного kubeconfig, можно ознакомиться на соответствующей странице документации.

4. **Prometheus-main** — сбор метрик провайдера dex.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `user-authn`

Подсистема Security

В данном подразделе описывается архитектура подсистемы Security (подсистемы безопасности) Deckhouse Kubernetes Platform (DKP).

В подсистему Security входят следующие модули:

- `admission-policy-engine` — позволяет использовать в кластере политики безопасности согласно Kubernetes Pod Security Standards. Для реализации политик модуль использует Gatekeeper;
- `runtime-audit-engine` — реализует рантайм-аудит (обнаружение угроз на основе событий ядра Linux и аудита Kubernetes API с помощью Falco). Архитектура описана в разделе «Рантайм-аудит»;
- `security-events-manager` — выполняет декларативный сбор, обработку, нормализацию и доставку событий безопасности из логов приложений и инфраструктурных компонентов Kubernetes. Архитектура описана в разделе «События безопасности»;
- `operator-trivy` — позволяет периодически сканировать кластер DKP на наличие уязвимостей;
- `cert-manager` — управляет TLS-сертификатами в кластере;
- `secrets-store-integration` — реализует доставку секретов в приложения Kubernetes-кластера путем подключения секретов, ключей и сертификатов из внешних хранилищ;
- `secret-copier` — автоматически копирует секреты в неймспейсы кластера.

Контроль целостности

Контроль целостности — совокупность механизмов проверки контейнеров для обеспечения безопасности и их соответствия заданной конфигурации.

В Deckhouse Kubernetes Platform (DKP) доступны следующие механизмы:

- контроль целостности пользовательской нагрузки при запуске — проверка подписи образов на этапе валидации запросов к API Kubernetes;
- контроль целостности работающих контейнеров — рантайм-аудит;
- защита целостности модулей DKP — установка модулей в виде неизменяемых образов.



Контроль целостности образов на уровне контейнерного рантайма (CRI) — проверка криптографической подписи образов и защита распакованных слоев от изменения — реализован только в редакциях DKP CSE Lite и CSE Pro. В остальных редакциях этот механизм недоступен (подробнее — в разделе «Контроль целостности образов на уровне CRI»).

Контроль целостности образов на уровне CRI

Проверка криптографической подписи образов при загрузке и старте контейнера, а также контроль неизменяемости распакованных слоев образа с помощью DM-Verity, реализованы на уровне containerd и доступны **только в редакциях DKP CSE Lite и CSE Pro**. Подписи образов

платформы проверяются встроенным в containerd набором публичных сертификатов; закрытый ключ подписи принадлежит поставщику платформы, добавить собственный ключ для проверки нельзя. Подробнее об этом механизме — в документации DKP CSE.

В остальных редакциях DKP (CE, BE, SE, SE+, EE) проверки подписи образов на уровне CRI нет:

- При использовании containerd v2 применяется снапшоттер EROFS: каждый слой OCI-образа преобразуется в отдельный файл в формате EROFS и монтируется только для чтения. Это снижает риск подмены данных в уже распакованных на узле образах, но не исключает его полностью и не заменяет проверку подлинности образа. Подробнее о переходе на containerd v2 — в разделе «Миграция container runtime на containerd v2».
- Проверка хеш-суммы SHA-256 при скачивании образа, выполняемая containerd, защищает от повреждения данных при передаче, но не подтверждает подлинность образа: она не позволяет определить, кем был собран образ.

Чтобы контролировать целостность и подлинность образов пользовательских приложений в этих редакциях, используйте проверку подписи на этапе валидации запросов к API Kubernetes (подробнее — в разделе «Контроль целостности пользовательской нагрузки при запуске»).

Контроль целостности пользовательской нагрузки при запуске

Целостность и подлинность образов пользовательских приложений проверяются на этапе валидации запросов к API Kubernetes — до создания пода, а не на уровне CRI. Механизм реализован модулем `admission-policy-engine` и использует подписи, созданные с помощью Cosign.

Последовательность контроля целостности при запуске:

1. На этапе сборки образ подписывается закрытым ключом с помощью Cosign. Подпись публикуется в хранилище образов контейнеров отдельным тегом и не изменяет сам образ.
2. В кластере создается ресурс `SecurityPolicy` с параметром `policies.verifyImageSignatures`, в котором указываются публичные ключи и шаблоны адресов образов.
3. При создании или обновлении пода валидационный вебхук передает запрос `Gatekeeper`, который обращается к компоненту `Ratify`.
4. `Ratify` проверяет подпись образа по указанным публичным ключам.
5. Если подпись отсутствует или не соответствует указанным ключам, создание пода запрещается.

Дополнительно можно ограничить перечень хранилищ образов контейнеров, из которых разрешен запуск подов, с помощью параметра `policies.allowedRepos` ресурса `OperationPolicy`.

Подробнее о настройке — в разделе «Проверка подписи образов».

Контроль целостности работающих контейнеров

Рантайм-аудит в DKP включает в себя анализ событий ядра Linux и аудита событий API Kubernetes. Это позволяет отслеживать, что приложения в подах работают в неизменном виде, соответствуют ожидаемому состоянию и не были модифицированы.

Для аудита используются:

- встроенные правила;
- пользовательские правила, которые можно добавлять с использованием синтаксиса условий `Falco`.

В процессе контроля целостности работающих контейнеров могут выявляться такие угрозы, как запуск оболочек командной строки в контейнерах или подах, обнаружение контейнеров, работающих в привилегированном режиме, монтирование небезопасных путей в контейнеры, попытки чтения секретных данных.

Подробнее о настройках рантайм-аудита можно почитать в разделе «Рантайм-аудит».

Защита целостности модулей DKP

Начиная с версии 1.74, модули DKP устанавливаются не в виде каталога с файлами, а в виде образа в формате EROFS, который подключается только для чтения через механизм ядра `DM-Verity`. Это защищает содержимое уже установленного модуля от изменения на узле: перезаписать файлы модуля в смонтированном образе невозможно, а `DM-Verity` проверяет блоки образа при обращении к ним.

Механизм включается автоматически, если на узле, где работает контроллер DKP (по умолчанию — `master`-узел), файловая система `erofs` зарегистрирована в ядре, то есть соответствующий модуль ядра загружен или встроен в ядро. Проверить это можно командой:

```
grep -w erofs /proc/filesystems
```

Если файловая система `erofs` недоступна, DKP продолжит работу и будет устанавливать модули обычным способом, без защиты их целостности. Переключение происходит без отдельного алерта, поэтому проверяйте доступность `erofs` заранее.



DKP загружает модуль ядра `erofs` и настраивает его автозагрузку только на узлах, где в качестве container runtime используется containerd v2. Если на master-узлах используется containerd v1, модуль `erofs` может оказаться незагруженным, и модули DKP будут установлены без защиты целостности — даже если ядро поддерживает `erofs`.

Чтобы гарантированно задействовать механизм, переведите master-узлы на containerd v2 либо обеспечьте загрузку модуля `erofs` средствами операционной системы.

При использовании этого механизма следует учитывать следующее:

- Механизм защищает модули от изменения после установки, но не подтверждает их подлинность: проверки криптографической подписи модулей нет. Источником доверия остается хранилище образов контейнеров, из которого модуль загружается.
- Механизм относится только к модулям DKP и не затрагивает образы пользовательских приложений.
- Механизм доступен во всех редакциях DKP и не требует отдельного включения.

Дополнительные ресурсы

- Кастомные ресурсы модуля `admission-policy-engine`

Архитектура рантайм-аудита

Рантайм-аудит Deckhouse Kubernetes Platform (DKP) основан на системе обнаружения угроз Falco. Этот механизм анализирует события ядра Linux и аудит событий Kubernetes API для выявления подозрительной активности в работающих контейнерах и кластере в целом.

DKP запускает объединённые в DaemonSet агенты Falco на каждом узле, после чего те приступают к сбору системных вызовов ОС и данных, полученных в ходе аудита событий Kubernetes.



Разработчики Falco рекомендуют запускать его как systemd-сервис, что может быть затруднительно в кластерах Kubernetes с поддержкой автомасштабирования. В DKP реализованы дополнительные механизмы безопасности, такие как мультитенантность и политики контроля ресурсов, которые в сочетании с использованием DaemonSet обеспечивают высокий уровень защиты.

На каждом узле кластера запускается под Falco со следующими компонентами:

- `falco` — собирает события, обогащает их метаданными и отправляет в stdout;
- `rules-loader` — собирает данные с правилами из кастомных ресурсов FalcoAuditRules и сохраняет их в общую директорию;
- `falcosidekick` — принимает события от `falco` и экспортирует их в виде метрик во внешние системы;
- `kube-rbac-proxy` — защищает эндпоинт метрик `falcosidekick` от неавторизованного доступа.

Подробнее с архитектурой модуля `runtime-audit-engine`, реализующего аудит событий безопасности DKP, можно ознакомиться в разделе с описанием архитектуры модуля.

Правила аудита

Для анализа событий безопасности применяются правила, определяющие критерии подозрительного поведения. Каждое правило представляет собой выражение, содержащее определённое условие и написанное в соответствии с синтаксисом условий Falco.

Встроенные правила

В DKP предусмотрены следующие виды встроенных правил:

- **правила для аудита Kubernetes**, которые помогают выявить проблемы с безопасностью DKP и самим механизмом аудита. Эти правила расположены в контейнере `falco` по пути `/etc/falco/k8s_audit_rules.yaml`;
- **нормативные правила**, удовлетворяющие требованиям приказа ФСТЭК России №118 от 4 июля 2022 г. «Требования по безопасности информации к средствам контейнеризации». Эти правила `fstec` описаны в формате кастомного ресурса `FalcoAuditRules`.

Пользовательские правила

Для добавления пользовательских правил используется кастомный ресурс `FalcoAuditRules`.

У каждого агента Falco есть сайдкар-контейнер с экземпляром сервиса `shell-operator`. Этот экземпляр считывает правила из ресурсов Kubernetes, конвертирует их в правила Falco и сохраняет правила в директорию `/etc/falco/rules.d/` в поде. При добавлении нового правила Falco автоматически обновляет конфигурацию.

Дополнительные ресурсы

- Документация модуля `runtime-audit-engine`

Архитектура событий безопасности

Модуль `security-events-manager` в составе в Deckhouse Kubernetes Platform (DKP) выполняет декларативный сбор, обработку, нормализацию и доставку событий безопасности, извлекаемых из логов приложений и инфраструктурных компонентов Kubernetes.

Событие безопасности — это структурированная запись о значимом с точки зрения информационной безопасности (ИБ) действии или факте. Типовые категории таких событий:

- аутентификация и авторизация;
- доступ к API и конфигурации;
- изменения объектов кластера;
- аномалии рантайма и сетевой активности.

Независимо от исходного формата лога, на выходе формируется единообразная модель события с обязательным минимумом атрибутов.



Модуль `security-events-manager` находится в стадии `Experimental`. Описанная архитектура может измениться в будущих версиях.

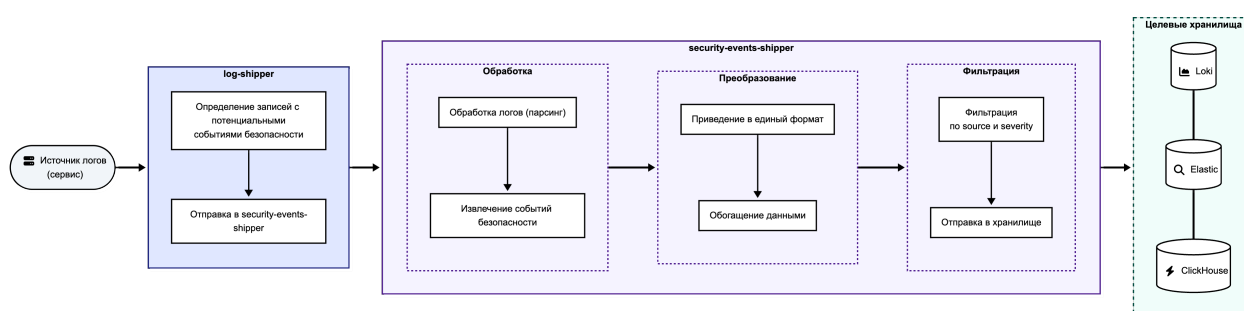
Архитектура решения

Архитектура предназначена для построения единого контура работы с событиями безопасности, извлекаемыми из логов приложений и инфраструктурных компонентов Kubernetes. Модуль выполняет три этапа обработки:

1. **Сбор** — сбор логов из подов и файлов на узлах, первичный отбор записей, потенциально содержащих события безопасности.
2. **Обработка и обогащение** — парсинг логов, извлечение признаков события, преобразование в единую модель и обогащение контекстом.
3. **Доставка** — фильтрация по политике и отправка в назначенные хранилища.

На уровне доставки поддерживается отправка в несколько типов систем хранения и аналитики (например, Loki, Elasticsearch, Kafka, Splunk, Vector, File).

Пайплайн обработки



Архитектура разделяет три фазы пайплайна: **сбор, обработка и обогащение, доставка**.

Слой сбора (log-shipper)

Сбор выполняется через вспомогательный модуль `log-shipper`:

- для контейнерных источников используются логи приложений в неймспейсе;
- для кластерных источников используются файлы на узлах и логи системных сервисов (например, `/var/log/kube-audit/audit.log`, `/var/log/auth.log`).

Применяется двухуровневая схема:

- модуль `log-shipper` выполняет предварительный отбор логовых записей с помощью простых операций сравнения (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) и передаёт их в шлюз;
- модуль `security-events-manager` (шлюз) выполняет парсинг полей, последующую обработку и отправку.

Поскольку парсинг логов — это ресурсоёмкая операция, он выполняется только для записей, заранее отобранных как потенциально содержащие события безопасности, а не для всех входящих логов. Поэтому на этапе первичного отбора не выполняется глубокая фильтрация по значениям полей, требующая полноценного парсинга содержимого.

Пример исходного лога для этапа отбора:

```
{
  "time": "2026-05-10T14:21:03Z",
  "kind": "Event",
  "source": "kube-apiserver",
  "level": "Metadata",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401,
  "requestURI": "/api/v1/namespaces/default/secrets",
  "user": "system:serviceaccount:default:demo",
  "sourceIPs": [
    "206.123.145.70"
  ]
}
```

Обработка и извлечение событий

После передачи в шлюз (gateway) выполняется распознавание структуры логов. Поддерживаются стандартные стратегии обработки:

- `JSON` — для структурированных логов;
- `Regex` — для строковых форматов с предсказуемым шаблоном;
- `Grok` — для сложных неунифицированных форматов.

Результат обработки используется для извлечения признаков события и построения унифицированного набора полей.

Пример результата этапа обработки:

```
{
  "parsed": {
    "timestamp": "2026-05-10T14:21:03Z",
    "source_component": "kube-apiserver",
    "http_status": 401,
    "request_uri": "/api/v1/namespaces/default/secrets",
    "actor_id": "system:serviceaccount:default:demo",
    "source_ip": "206.123.145.70"
  }
}
```

После обработки запись становится источником полей для классификации события (код/категория/критичность (severity)/результат (outcome)), а также для построения контекста

(`actor`, сетевые атрибуты, метаданные источника).

Преобразование и обогащение события

Преобразование выполняется в два этапа:

1. **Transform** — сопоставление полей исходного лога с полями целевой модели события.
2. **Enrich** — добавление или уточнение полей из дополнительных источников контекста (например, статических атрибутов среды, роли субъекта, служебных признаков).

Порядок фиксирован: сначала применяется `Transform`, затем `Enrich`. При конфликте целевого поля итоговое значение определяется этапом `Enrich`.

Фильтрация и доставка событий

После формирования события применяется политика доставки:

- фильтрация по источникам;
- фильтрация по минимальному уровню критичности;
- маршрутизация в одно или несколько назначений.

Правила фильтрации могут использовать как точные идентификаторы источников, так и маски источников, что позволяет управлять потоками на уровне отдельных сервисов или целых групп.

В качестве назначений используются системы хранения и обработки событий: кластерный Loki, внешние SIEM/лог-платформы и потоковые шины. Схема доставки поддерживает параллельную отправку в несколько целевых систем.

Модель события безопасности

Ниже приведена структура события, которое отправляется в хранилище.

Обязательные поля:

- `id`, `timestamp`;
- `source.component`;
- `event.code`, `event.category`, `event.severity`, `event.outcome`;
- `eventMetadata.cluster`.

Опциональные поля:

- `eventMetadata.sourceIPs`;
- `actor.*`;
- `object.*`.

```
{
  "id": "2f0de5c2-2e58-4d3f-b4fe-5ec6f1935b9f",
  "timestamp": "2026-05-10T14:21:03Z",
  "source": {
    "component": "kube-apiserver"
  },
  "event": {
    "code": "UNAUTHORIZED_ACCESS",
    "category": "Rbac",
    "severity": "High",
    "outcome": "Failure"
  },
  "eventMetadata": {
    "cluster": "prod-cluster",
    "sourceIPs": [
      "206.123.145.70"
    ]
  },
  "actor": {
    "id": "system:serviceaccount:default:demo",
    "type": "ServiceAccount"
  },
  "object": {
    "id": "/api/v1/namespaces/default/secrets",
    "type": "KubernetesResource"
  }
}
```

Минимально достаточный цикл работы

В практическом сценарии архитектура работает по следующей цепочке:

1. Источники логов подключаются к контуру сбора.
2. Выполняется первичный отбор потенциально релевантных записей.
3. Записи обрабатываются и преобразуются в унифицированные события.
4. События обогащаются контекстными атрибутами.
5. Применяются правила фильтрации и маршрутизации.
6. События отправляются в целевые хранилища и аналитические системы.

Итогом является единый и управляемый поток событий безопасности, пригодный для мониторинга, анализа и долгосрочного аудита.

Поддерживаемые события безопасности

Модуль `security-events-manager` поставляется со встроенным набором правил обнаружения событий безопасности, охватывающим аутентификацию, конфигурацию, RBAC, среду выполнения и другие категории. Актуальный перечень реализованных событий безопасности, их коды, критичность и описание приведены в документации модуля.

Дополнительные ресурсы

- Документация модуля `log-shipper`
- Документация модуля `security-events-manager`

Модуль admission-policy-engine

Модуль `admission-policy-engine` обеспечивает применение политик безопасности и операционных политик в кластере Kubernetes. Политики применяются на основе проверок по Pod Security Standards и правил из следующих кастомных ресурсов:

- `OperationPolicy` — описывает операционную политику кластера;
- `SecurityPolicy` — описывает политику безопасности кластера;
- `SecurityPolicyException` — описывает исключения из политики безопасности кластера.

i Обработка ресурсов OperationPolicy и SecurityPolicy выполняется модулем `deckhouse`. Deckhouse-контроллер модуля `deckhouse` с использованием `addon-operator` и хуков модуля на основе OperationPolicy и SecurityPolicy создает кастомные ресурсы для Gatekeeper. Gatekeeper использует созданные кастомные ресурсы для валидации создаваемых или обновляемых ресурсов Kubernetes.

Подробнее о концепции хуков можно узнать из документации `addon-operator`.

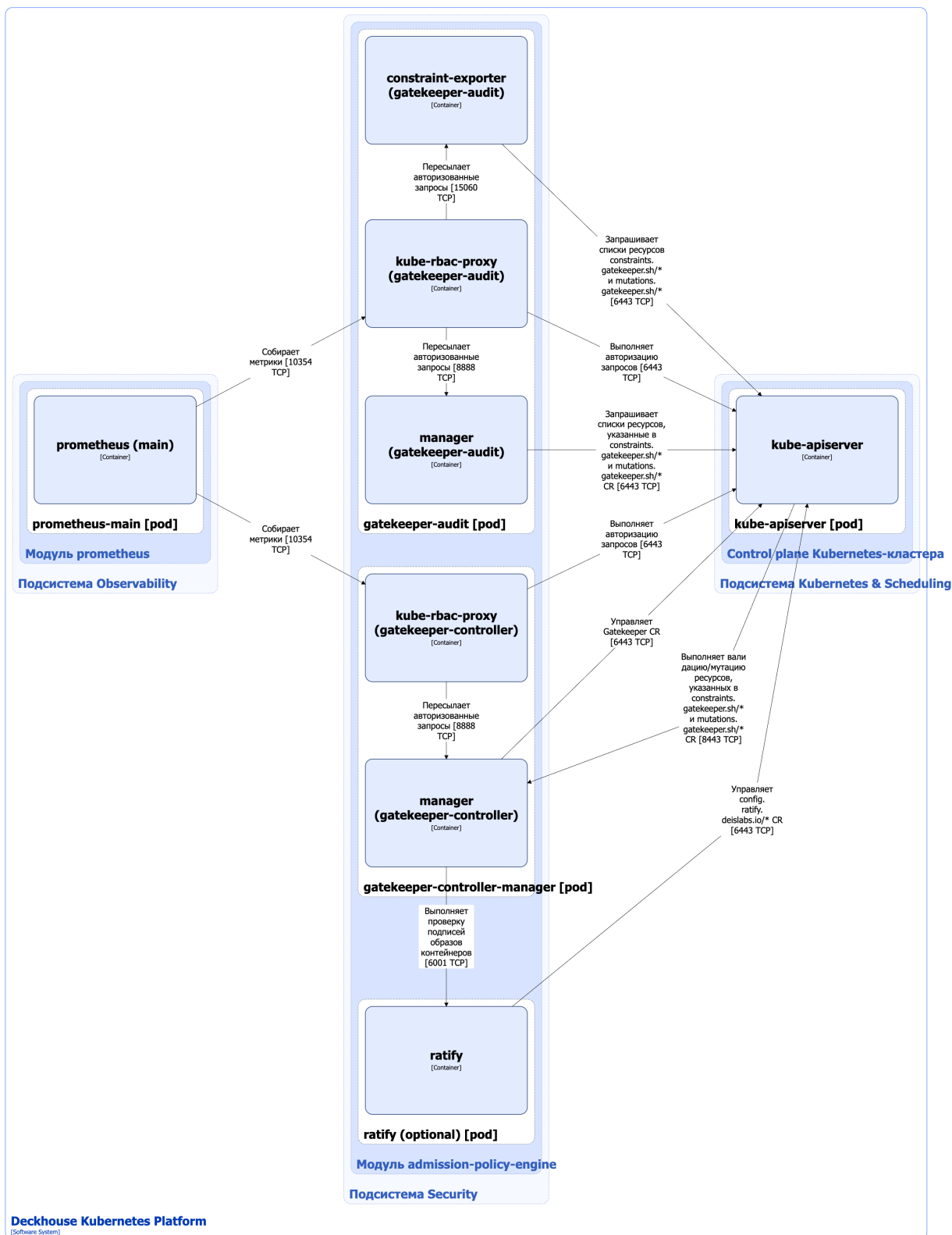
Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

Архитектура модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `admission-policy-engine` на уровне 2 модели C4 и его взаимодействие с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Gatekeeper-controller-manager** — это контроллер (Gatekeeper), выполняющий следующие операции:
 - управление кастомными ресурсами Gatekeeper;
 - валидация ресурсов Kubernetes, указанных в кастомных ресурсах из `constraints.gatekeeper.sh/*` API-группы;
 - мутация ресурсов Kubernetes, указанных в кастомных ресурсах `AssignMetadata`, `Assign`, `ModifySet` и `AssignImage`.

Правила безопасности задаются с помощью кастомных ресурсов `ConstraintTemplate` и

кастомных ресурсов из API-группы `constraints.gatekeeper.sh/*`. `ConstraintTemplate` описывает новые типы политик, на основании которых создаются конкретные политики безопасности для проверки ресурсов.

Состоит из следующих контейнеров:

- **manager** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера.
2. **Gatekeeper-audit** — реализует функционал периодической проверки существующих ресурсов Kubernetes на соответствие политикам безопасности.

Состоит из следующих контейнеров:

- **manager** — основной контейнер;
 - **constraint-exporter** — сайдкар-контейнер, предоставляющий дополнительные метрики по кастомным ресурсам `constraints.gatekeeper.sh/*` и `mutations.gatekeeper.sh/*`;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам `manager` и `constraint-exporter`.
3. **Ratify** — опциональный компонент, состоящий из одного контейнера **ratify**. Он представляет собой реализацию провайдера Gatekeeper для проверки метаданных используемых артефактов. В DKP этот провайдер применяется для проверки подписи образов контейнеров и доступен в редакциях DKP SE+, EE, CSE Lite и CSE Pro.

i

Gatekeeper использует кастомный ресурс Provider для расширения функционала по валидации ресурсов Kubernetes. Ресурс Provider описывает эндпоинт сервиса, на который Gatekeeper передает запрос при выполнении ValidationWebhook. Некоторые модули DKP, такие как `operator-trivy`, могут создавать кастомные ресурсы Provider и тем самым расширять функционал проверок.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

- **Kube-apiserver:**
 - мониторинг ресурсов Kubernetes, указанных в кастомных ресурсах из API-групп `constraints.gatekeeper.sh/*` и `mutations.gatekeeper.sh/*`;
 - работа с кастомными ресурсами `ConstraintTemplate`, `Assign`, `AssignImage`, `AssignMetadata`, `ModifySet`, а также с ресурсами из API-групп `constraints.gatekeeper.sh/*` и `config.ratify.deislabs.io/*`.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — валидация ресурсов Kubernetes и проверка на соответствие заданным правилам безопасности.
2. **Prometheus-main** — сбор метрик модуля.

Дополнительные ресурсы

- Документация модуля `admission-policy-engine`
- Документация модуля `deckhouse`

Модуль cert-manager

Модуль `cert-manager` автоматизирует полный цикл управления сертификатами в кластере: от выпуска и продления сертификатов, включая самоподписанные, до интеграции с внешними центрами сертификации, такими как Let's Encrypt, HashiCorp Vault и Venafi. Это упрощает обеспечение безопасности сервисов и позволяет централизованно контролировать процессы, связанные с сертификатами.

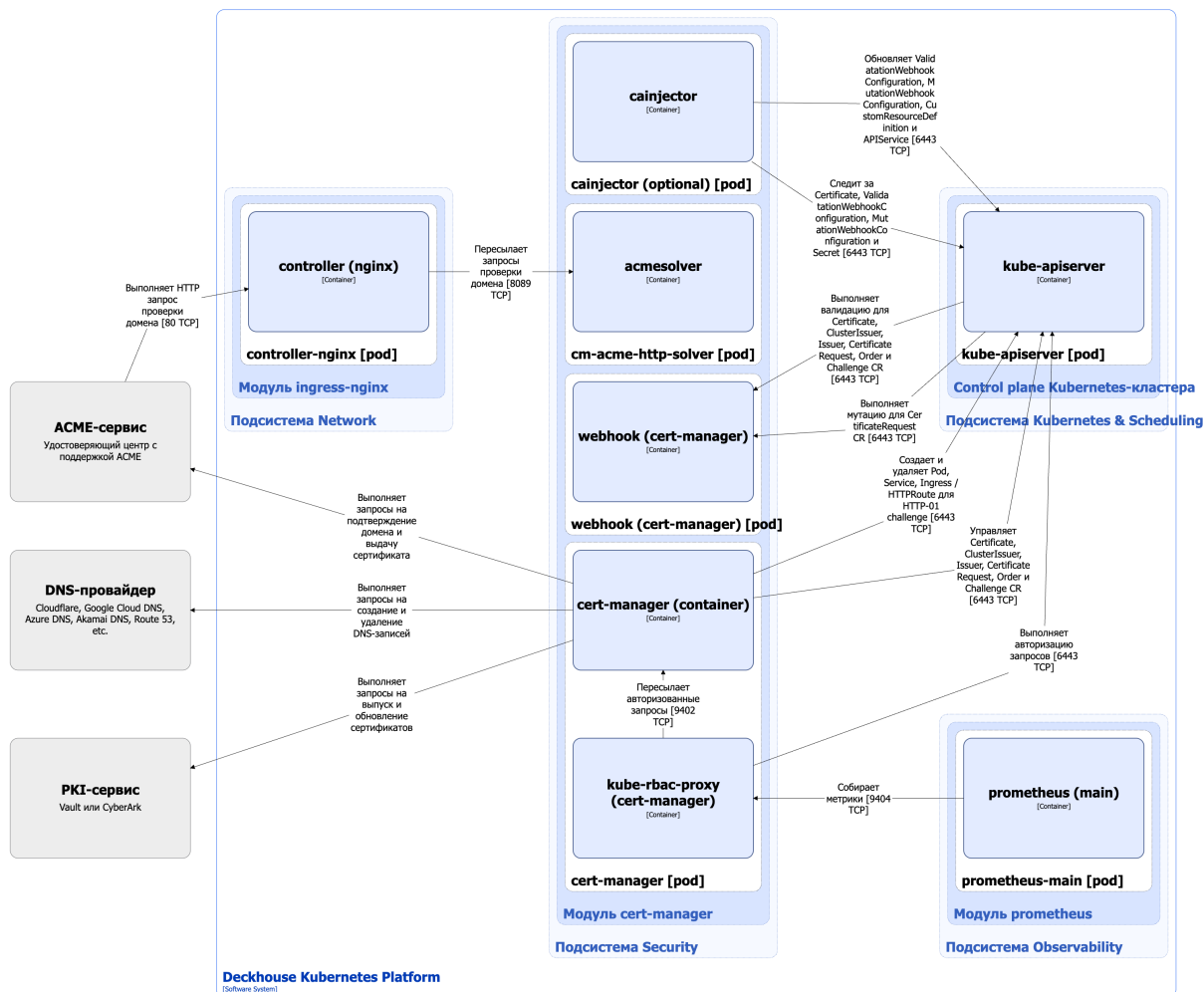
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `cert-manager` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты модуля

Модуль `cert-manager` состоит из следующих компонентов:

1. **Cert-manager** — контроллер, обеспечивающий полный цикл управления сертификатами в Deckhouse Kubernetes Platform (DKP). `Cert-manager` управляет следующими кастомными ресурсами:
 - `Issuer` — описывает настройки источника, от которого выпускаются сертификаты, например CA, ACME-сервис или внешний PKI-провайдер. Действует в пределах выбранного неймспейса;
 - `ClusterIssuer` — кластерный аналог `Issuer`: действует на весь кластер и доступен во всех неймспейсах;
 - `Certificate` — описывает требуемый сертификат: субъект, срок действия, используемый `Issuer` или `ClusterIssuer`, а также дополнительные параметры выпуска;
 - `CertificateRequest` — заявка на выпуск или продление сертификата;
 - `Challenge` — задание для прохождения проверки владения доменом, например с помощью HTTP-01 или DNS-01 challenge в ACME;
 - `Order` — объект ACME, объединяющий связанные `Challenge` в последовательность для получения сертификата у ACME-сервера, например Let's Encrypt.

Компонент содержит следующие контейнеры:

- **cert-manager** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера cert-manager.

i

Для прохождения DNS-01 challenge cert-manager поддерживает ряд популярных DNS-провайдеров, включая AzureDNS, Cloudflare и DigitalOcean. С полным списком поддерживаемых DNS-провайдеров можно ознакомиться в документации cert-manager. Для провайдеров, не поддерживаемых «из коробки», используются issuer типа вебхук. Такие issuer являются внешними компонентами, поэтому для корректной работы их необходимо устанавливать в системный неймспейс d8-cert-manager. При внесении изменений в этом неймспейсе учитывайте наличие подобных расширений, чтобы сохранить их работоспособность.

2. **Webhook** — компонент, состоящий из одного вебхук-контейнера и выполняющий следующие операции:
 - валидацию кастомных ресурсов Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, Order;
 - мутацию кастомных ресурсов CertificateRequest — вебхук добавляет информацию о пользователе, создавшем запрос на сертификат.

В DKP валидация отключена для ресурсов в неймспейсе d8-cert-manager, а также для неймспейсов с лейблом cert-manager.io/disable-validation=true.

3. **Cainjector** — дополнительный компонент, состоящий из одного контейнера cainjector. Cainjector отвечает за автоматическую подстановку или обновление сертификатов корневого центра сертификации (CA) во все релевантные ресурсы Kubernetes: ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition и APIService. Это обеспечивает актуальность доверенных корневых сертификатов для сервисов, использующих вебхуки и расширения API.

Cainjector включается параметром .spec.settings.enableCAInjector в настройках модуля cert-manager. DKP не использует cainjector для собственных компонентов, поэтому включайте его только в том случае, если вашим приложениям или внешним расширениям требуется инъекция CA в ресурсы Kubernetes.

Cainjector обрабатывает только ресурсы с аннотациями cert-manager.io/inject-ca-from, cert-manager.io/inject-ca-from-secret или cert-manager.io/inject-apiserver-ca в зависимости от типа ресурса.

4. **Cm-acme-http-solver** — временный под с контейнером acmesolver, запускаемый для прохождения HTTP-01 Challenge при валидации домена через ACME (например, Let's Encrypt). Этот под автоматически создаётся контроллером cert-manager на время исполнения HTTP-01 Challenge и удаляется по завершении процедуры. Такой подход реализует безопасную временную публикацию ресурса, подтверждающего владение доменом для получения сертификата.

Взаимодействия модуля

Модуль cert-manager взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - управляет кастомными ресурсами Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, Order;
 - отслеживает и обновляет ресурсы ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition и APIService.
2. **ACME-сервис** — обрабатывает запросы на подтверждение домена и выпуск сертификатов.
3. **PKI-сервис** — обрабатывает запросы для выпуска и обновления (перевыпуска) сертификатов.
4. **DNS-провайдер** — обрабатывает запросы на добавление и удаление записей в службах DNS для прохождения DNS-01 Challenge при валидации домена через ACME-сервер.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver:**
 - отправляет вебхук-запросы на валидацию кастомных ресурсов Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, Order;

- отправляет вебхук-запросы на мутацию кастомных ресурсов CertificateRequest.
2. **Prometheus-main** — собирает метрики `cert-manager`.
 3. **Nginx Controller** — пересылает запросы от ACME-сервера к `cm-acme-http-solver`.

Дополнительные ресурсы

- Документация модуля `cert-manager`

Модуль operator-trivy

Модуль `operator-trivy` обеспечивает сканирование пользовательских образов в рантайм на известные CVE (Common Vulnerabilities and Exposures), включая уязвимости Astra Linux, ALT Linux и РЕД ОС. Базируется на проекте Trivy. Для сканирования используются публичные базы уязвимостей, обогащаемые базами Astra Linux, ALT Linux и РЕД ОС, а также БДУ ФСТЭК (Банком данных угроз Федеральной службы по техническому и экспортному контролю).

Также модуль производит анализ соответствия кластера Kubernetes требованиям CIS (Center for Internet Security) Kubernetes Benchmark.

Дополнительно модуль `operator-trivy` обеспечивает возможность собирать поведенческие модели рабочей нагрузки при использовании опционального компонента `node-agent`.

Модуль `operator-trivy` работает с кастомными ресурсами API-групп `trivy.deckhouse.io` и `spdx.softwarecomposition.kubescape.io`.

В API-группу `trivy.deckhouse.io` входят следующие ресурсы:

- `ClusterComplianceReport` — хранит сводный отчёт о соответствии кластера требованиям безопасности (например, CIS Kubernetes Benchmark);
- `ClusterConfigAuditReport` — хранит результаты аудита конфигурации Kubernetes-объектов на уровне кластера;
- `ClusterInfraAssessmentReport` — хранит результаты проверок безопасности инфраструктуры Kubernetes на уровне кластера;
- `ClusterRbacAssessmentReport` — хранит результаты проверки настроек управления доступом на основе ролей (RBAC) на уровне кластера;
- `ClusterSbomReport` — хранит сводный SBOM (Software Bill of Materials) по программным компонентам на уровне кластера;
- `ClusterVulnerabilityReport` — хранит агрегированные результаты сканирования уязвимостей на уровне кластера;
- `ConfigAuditReport` — хранит результаты аудита конфигурации Kubernetes-объектов;
- `ExposedSecretReport` — хранит результаты поиска потенциальных секретов в образе контейнера;
- `InfraAssessmentReport` — хранит результаты проверок безопасности инфраструктуры Kubernetes-объектов;
- `NodeVulnerabilityReport` — хранит результаты сканирования уязвимостей в `rootfs` (корневой файловой системе) узла;
- `RbacAssessmentReport` — хранит результаты проверки RBAC-настроек на предмет избыточных привилегий и других рисков;
- `RegistryImageVulnerabilityReport` — хранит результаты CVE-сканирования конкретного тега образа из реестра;
- `RegistryScanTarget` — задаёт целевой реестр, репозитории и параметры периодического сканирования;
- `SbomReport` — хранит SBOM (Software Bill of Materials), то есть состав ПО и зависимостей в образе контейнера;
- `VulnerabilityReport` — хранит результаты сканирования уязвимостей в образе контейнера.

В API-группу `spdx.softwarecomposition.kubescape.io` входят следующие ресурсы:

- `ApplicationProfiles` — хранит профиль поведения приложения в рантайме (системные вызовы, запускаемые процессы, доступ к файлам, HTTP-эндпоинты);
- `CollapseConfigurations` — задаёт параметры объединения динамических путей, эндпоинтов и сетевых адресов для уменьшения объёма рантайм-профилей;
- `ConfigurationScanSummaries` — хранит сводку результатов проверок конфигурации для группы рабочих нагрузок в заданной области (например, неймспейсе);
- `ContainerProfiles` — хранит профиль отдельного контейнера, включая рантайм-поведение и сетевые взаимодействия;
- `GeneratedNetworkPolicies` — хранит сгенерированные на основе наблюдаемого трафика объекты `NetworkPolicy`;
- `KnownServers` — хранит справочник известных серверов и диапазонов IP-адресов для обогащения сгенерированных сетевых политик;
- `NetworkNeighborhoods` — хранит наблюдаемую карту входящих и исходящих сетевых взаимодействий рабочих нагрузок;

- OpenVulnerabilityExchangeContainers — хранит документы формата OpenVEX со статусами уязвимостей для компонентов;
- SbomSyftFiltered — хранит отфильтрованный Syft SBOM только с релевантными уязвимыми компонентами;
- SbomSyfts — хранит SBOM в формате Syft;
- SeccompProfiles — хранит seccomp-профили контейнеров (разрешённые системные вызовы и правила фильтрации);
- VulnerabilityManifestSummaries — хранит сводку VulnerabilityManifest по уровням критичности с ссылками на связанные объекты;
- VulnerabilityManifests — хранит подробный манифест найденных уязвимостей;
- VulnerabilitySummaries — хранит агрегированную сводку уязвимостей для заданной области видимости;
- WorkloadConfigurationScans — хранит детальные результаты сканирования конфигурации конкретного workload;
- WorkloadConfigurationScanSummaries — хранит сводку результатов WorkloadConfigurationScan.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `operator-trivy` на уровне 2 модели C4 и его взаимодействие с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Operator** (Deployment) — компонент, реализованный на базе Trivy Operator, следит за ресурсами Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob и Job и при изменении их спецификаций запускает задачу Job для выполнения сканирования.

Operator анализирует результаты работы Job и формирует следующие отчёты в виде кастомных ресурсов:

- ClusterComplianceReport;
- ClusterConfigAuditReport;
- ClusterInfraAssessmentReport;
- ClusterRbacAssessmentReport;
- ClusterVulnerabilityReport;
- ConfigAuditReport;
- ExposedSecretReport;
- InfraAssessmentReport;
- NodeVulnerabilityReport;
- RbacAssessmentReport;
- SbomReport;
- VulnerabilityReport.

Состоит из следующих контейнеров:

- **operator** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам контроллера.

2. **Trivy-server** (StatefulSet) — компонент реализует сервис сканирования безопасности, базируется на основе Open Source-проекта Trivy.

Trivy-server при старте и впоследствии регулярно обновляет базу данных уязвимостей из образа формата Open Container Initiative (OCI).

Trivy-server обрабатывает запросы от других компонентов модуля (operator, registry-scanner, trivy-provider, задачи сканирования), выполняет запрошенное сканирование и возвращает результат.

Состоит из следующих контейнеров:

- **server** — основной контейнер;
- **trivy-db-info** — сайдкар-контейнер, который синхронизирует метаданные кеша Trivy с ресурсом ConfigMap `trivy-db-info` в неймспейсе `d8-operator-trivy`.

3. **Trivy-provider** (StatefulSet) — компонент состоит из одного контейнера **trivy-provider** и предоставляет интерфейс для проверки образов компонентом Gatekeeper модуля `admission-policy-engine`.

При установке модуля создаётся ресурс Provider, который регистрирует trivy-provider как провайдер в Gatekeeper. Подробнее с описанием интеграции можно ознакомиться в соответствующем разделе документации Gatekeeper.

Deckhouse-контроллер разворачивает этот компонент, если параметр `.settings.denyVulnerableImages.enabled` кастомного ресурса ModuleConfig принимает значение `true` (по умолчанию — `false`) и включён компонент Gatekeeper модуля `admission-policy-engine`.

4. **Report-updater** (Deployment) — опциональный контроллер, состоящий из одного контейнера **report-updater**, реализует MutatingWebhook и обеспечивает обогащение кастомного ресурса VulnerabilityReport идентификаторами БДУ. Словарь БДУ регулярно (каждые 6 часов) обновляется из OCI-образа.

Deckhouse-контроллер разворачивает этот компонент, если параметр `.settings.linkCVEtoBDU` кастомного ресурса ModuleConfig принимает значение `true` (по умолчанию — `false`).

5. **Registry-scanner** (StatefulSet) — опциональный компонент, состоящий из одного контейнера **registry-scanner**, выполняет регулярное сканирование образов, размещённых в произвольных реестрах контейнеров, указанных пользователем. Сканирование нагрузок в самом кластере выполняется компонентом operator.

Deckhouse-контроллер разворачивает этот компонент, если параметр `.settings.registryScanning.enabled` кастомного ресурса ModuleConfig принимает значение `true` (по умолчанию — `false`).

Registry-scanner читает кастомные ресурсы RegistryScanTarget, запускает сканирование образов через компонент trivy-server и сохраняет результаты обработки в кастомном ресурсе RegistryImageVulnerabilityReport.

6. **Security-storage** (Deployment) — контроллер, состоящий из одного контейнера **apiserver**, является расширением API Kubernetes и выполняет обработку запросов к кастомным ресурсам API-групп `trivy.deckhouse.io` и `spdx.softwarecomposition.kubescape.io`.

Также security-storage реализует бэкенд для хранения этих ресурсов:

- метаданные сохраняются в базе данных SQLite;
- тело объектов сохраняется в gob-формате в каталоге `/data`.

7. **Node-agent** (DaemonSet) — опциональный компонент с одним контейнером **node-agent**, который запускается на всех узлах кластера в привилегированном режиме, и, используя eBPF-подпрограммы, наблюдает за поведением контейнеров и формирует рантайм-профили.

Node-agent сохраняет рантайм-профили в кастомных ресурсах ApplicationProfile и NetworkNeighborhood (в компоненте security-storage). Подробнее с работой node-agent можно ознакомиться в разделе документации модуля.



У компонента `node-agent` есть привилегированный доступ к операционной системе каждого узла. В Linux для этого требуются `capabilities`:

- `SYS_ADMIN`
- `SYS_PTRACE`
- `NET_ADMIN`
- `SYSLOG`
- `SYS_RESOURCE`
- `IPC_LOCK`
- `NET_RAW`

Это необходимо для наблюдения за поведением контейнеров, используя eBPF-подпрограммы.

`Node-agent` работает в профилирующем режиме, не выполняет активное обнаружение атак по правилам.

Deckhouse-контроллер разворачивает этот компонент, если параметр `.settings.nodeAgent.enabled` кастомного ресурса `ModuleConfig` принимает значение `true` (по умолчанию — `false`).

8. **Scan-noderootfs-<HASH>** (Job) — компонент, состоящий из одного контейнера **node-rootfs-scanner**, реализует задачу по сканированию корневой файловой системы (`scan-noderootfs`) узла кластера. Задача создаётся и управляется компонентом `operator`.
9. **Scan-vulnerabilityreport-<HASH>** (Job) — компонент обеспечивает запуск задач по сканированию безопасности Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob и Job с использованием компонента `trivy-server`. Задача создаётся и управляется компонентом `operator`.

Состоит из набора контейнеров **<CONTAINER_NAME>**, каждый из которых отвечает за сканирование соответствующего контейнера рабочей нагрузки (`workload`). В качестве базового образа для них используется образ `trivy` с добавленным `trivy-wrapper`, а в аргументах для него передаётся образ, заданный в спецификации целевого контейнера. `Trivy-wrapper` выполняет авторизацию у хранилища образов командой `trivy registry login`, а после передаёт управление `trivy`.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - авторизация запросов к метрикам `operator`;
 - работа с кастомными ресурсами API-группы `trivy.deckhouse.io`;
 - работа с кастомными ресурсами `RegistryScanTarget` и `RegistryImageVulnerabilityReport`;
 - отслеживание ресурсов Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob и Job;
 - создание, обновление и удаление ресурсов `Secret`, `ConfigMap` и `Job`.
2. **Registry** — загрузка OCI-образов с базой уязвимостей и базой БДУ.
3. **Хранилище образов** — сканирование образов

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver**:
 - пересылка API-запросов по кастомным ресурсам из API-групп `trivy.deckhouse.io` и `spdx.softwarecomposition.kubescape.io`;
 - мутация кастомных ресурсов `VulnerabilityReport`.
2. **Prometheus-main** — сбор метрик модуля.

Дополнительные ресурсы

- Документация модуля `admission-policy-engine`
- Документация модуля `operator-trivy`
- Документация модуля `registry`

Модуль runtime-audit-engine

Модуль `runtime-audit-engine` реализует в Deckhouse Kubernetes Platform (DKP) рантайм-аудит, основанный на системе обнаружения угроз Falco. Модуль собирает события ядра Linux и события аудита API Kubernetes (с помощью плагина `k8saudit`), обогащает их метаданными о подах Kubernetes и формирует события безопасности по установленным правилам. Правила аудита определяются с использованием кастомного ресурса `FalcoAuditRules`.

При включении модуля в неймспейсе `d8-runtime-audit-engine` создаётся ConfigMap `control-plane-configurator` с URL и CA для `audit webhook`. Модуль `control-plane-manager` обнаруживает этот ConfigMap и настраивает `control plane` на отправку событий аудита API Kubernetes в модуль `runtime-audit-engine`.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `runtime-audit-engine` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль `runtime-audit-engine` состоит из следующих компонентов:

1. **Runtime-audit-engine** (DaemonSet) — компонент, запускаемый на каждом узле кластера. Он собирает события аудита, проводит проверку правил и выводит сработавшие правила в `stdout`, а также экспортирует их как метрики в формате Prometheus. Данные поступают из ядра Linux через перехват системных вызовов (`syscalls`), а также из `containerd` через Unix-сокеты.

Компонент содержит следующие контейнеры:

- **falco** — основной контейнер, обеспечивающий сбор событий безопасности с узлов кластера и с контейнерных приложений в DKP на основе системы обнаружения угроз Falco;
- **falcosidekick** — сайдкар-контейнер, получающий события от компонента `falco` и экспортирующий события аудита в формате метрик Prometheus;
- **rules-loader** — сайдкар-контейнер, выполняющий следующие действия:
 - отслеживает кастомные ресурсы `FalcoAuditRules` и сохраняет их в общий каталог пода `/etc/falco/rules.d/` для обработки компонентом `falco`;
 - валидирует кастомный ресурс `FalcoAuditRules`;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC (Role-Based Access Control) для организации защищённого доступа к метрикам компонента.



Контейнер `falco` имеет привилегированный доступ к операционной системе каждого узла. Контекст безопасности контейнера включает capabilities `BPF`, `SYS_RESOURCE`, `PERFMON`, `SYS_PTRACE` и `SYS_ADMIN`.

2. **K8s-metacollector** (Deployment) — компонент, выполняющий проксирование запросов к `kube-apiserver` для уменьшения нагрузки на `control plane`. Также компонент уменьшает объём метаданных, передаваемых в `falco`, оставляя только данные, относящиеся к узлу. `K8s-metacollector` собирает метаданные из `kube-apiserver` о Pod, Namespace, Deployment, ReplicaSet, ReplicationController и Service.

Компонент содержит следующие контейнеры:

- **k8s-metacollector** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам `k8s-metacollector`.

Взаимодействия модуля

Модуль `runtime-audit-engine` взаимодействует со следующими компонентами:

1. Kube-apiserver:

- о работа с кастомными ресурсами `FalcoAuditRules`;
- о мониторинг ресурсов `Pod`, `Namespace`, `Deployment`, `ReplicaSet`, `ReplicationController` и `Service`;
- о авторизация запросов компонентов модуля.

2. Containerd:

- о получение метаданных контейнеров;
- о отслеживание событий `containerd`.

3. Ядро Linux — перехват системных вызовов (syscalls) из ядра Linux в режиме реального времени.

С модулем взаимодействуют следующие внешние компоненты:

1. Kube-apiserver:

- о отправляет вебхук-запросы на валидацию кастомного ресурса `FalcoAuditRules`;
- о отправляет события аудита.

2. Prometheus-main — собирает метрики модуля.

Дополнительные ресурсы

- Документация модуля `control-plane-manager`
- Документация модуля `runtime-audit-engine`

Модуль secrets-store-integration

Модуль `secrets-store-integration` реализует доставку секретов для приложений в Deckhouse Kubernetes Platform (DKP) из внешнего хранилища, совместимого с API HashiCorp Vault.

Модуль предоставляет следующие возможности:

- доставка секретов в поды в виде примонтированных файлов или переменных окружения без хранения в `etcd`;
- автоматическое подключение к внутреннему экземпляру Deckhouse Stronghold без ручной настройки (режим `DiscoverLocalStronghold`);
- работа с любым Vault-совместимым хранилищем секретов в режиме `Manual`;
- автоматическое обновление примонтированных секретов каждые две минуты при изменении значения в хранилище;
- подмена команды запуска (`entrypoint`) для приложений, которые нельзя модифицировать для прямого чтения секретов из хранилища;
- доставка бинарных секретов в формате Base64 (например, JKS-хранилища, keytab-файлы для Kerberos) с автоматическим раскодированием.

Режим работы (`Manual` или `DiscoverLocalStronghold`) задаётся параметром `settings.connectionConfiguration` в настройках модуля.

Модуль работает со следующими кастомными ресурсами:

- `SecretProviderClass` — описывает, какие секреты и из какого внешнего хранилища нужно доставлять в под. В спецификации этого ресурса также определяются параметры подключения к источнику секретов и сопоставление путей в контейнере;
- `SecretProviderClassPodStatus` — содержит статус процесса монтирования секретов в под и диагностическую информацию;
- `SecretsStoreImport` — хранит сопоставление секретов между Vault-совместимым хранилищем и файлами в контейнерах.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `secrets-store-integration` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль `secrets-store-integration` состоит из следующих компонентов:

1. **Csi-secrets-store** (DaemonSet) — компонент работает на всех узлах кластера и реализует стандарт Container Storage Interface (CSI) для обеспечения доставки секретов в под в виде примонтированных файлов.

Csi-secrets-store выполняет следующие действия:

- регистрирует эфемерный CSI-драйвер `secrets-store.csi.deckhouse.io` в kubelet;
- взаимодействует с `vault-csi-provider` для получения данных из хранилища секретов;
- монтирует секреты в виде файлов в под;
- выполняет ротацию секретов;
- отслеживает ресурсы Pod, CSIDriver и кастомный ресурс `SecretProviderClass`;
- работает с кастомным ресурсом `SecretProviderClassPodStatus`.

Компонент содержит следующие контейнеры:

- **injector-puller** — init-контейнер, который выполняет однократный запуск файл-инжектора (исполняемого файла для безопасного внедрения секретов из внешнего хранилища в окружение приложения внутри пода) из служебного образа `secrets-store-integration/env-injector`. Однократный запуск необходим для предварительной загрузки этого образа на каждый узел кластера. Образ используется компонентом webhook для доставки секретов в пользовательское приложение в виде переменных окружения.
 - **csi-node-driver-registrar** — сайдкар-контейнер, регистрирующий CSI Node Plugin в kubelet. Вызывает RPC `GetPluginInfo` и `NodeGetInfo` в контейнере `secrets-store`, чтобы получить информацию о плагине и узле;
 - **secrets-store** — основной контейнер;
 - **csi-livenessprobe** — сайдкар-контейнер, который отслеживает состояние Unix-сокета CSI-драйвера и предоставляет HTTP-эндпоинт `/healthz`, за которым следит kubelet. При неуспешном выполнении проверки `livenessProbe` kubelet перезапускает под `csi-secrets-store`;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам контейнера `secrets-store`.
2. **Vault-csi-provider** (DaemonSet) — компонент работает на всех узлах кластера и состоит из одного контейнера **vault-csi-provider**. Vault-csi-provider авторизуется в хранилище секретов, получает из него данные и передаёт их csi-secrets-store.
 3. **Ssi-controller** (Deployment) — контроллер состоит из одного контейнера **ssi-controller**. Контроллер отслеживает кастомные ресурсы `SecretsStoreImport` и создаёт на их основе кастомные ресурсы `SecretProviderClass`.
 4. **Webhook** (Deployment) — компонент реализует mutating-вебхуки для ресурсов Pod через механизм `Mutating Admission Controllers`.

Компонент изменяет манифест пода, если задана аннотация пода `secrets-store.deckhouse.io/role`. При этом компонент выполняет следующие модификации манифеста пода:

- добавляет init-контейнер, который копирует из служебного образа `secrets-store-integration/env-injector` статически собранный файл-инжектор во временную директорию, общую для всех контейнеров пода;
- если в манифесте пода задана аннотация `secrets-store.deckhouse.io/env-from-path` или контейнер использует секреты из хранилища, то для каждого такого

контейнера, включая init-контейнеры, компонент заменяет оригинальную команду запуска на запуск файл-инжектора.

В качестве аргумента файл-инжектор получает оригинальную команду запуска. Если в манифесте пода у контейнера отсутствует команда запуска, компонент получает команду запуска из образа в хранилище образов контейнеров.

Компонент содержит следующие контейнеры:

- **vault-secrets-webhook** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам контейнера vault-secrets-webhook.

Следующие компоненты не входят в состав модуля, но модуль влияет на них:

- **User-app** — компонент представляет типовое приложение пользователя, в которое требуется доставить секреты в переменные окружения.

Компонент содержит следующие контейнеры:

- **copy-env-injector** — init-контейнер, добавленный компонентом webhook, который выполняет копирование исполняемого файл-инжектора из служебного образа `secrets-store-integration/env-injector`;
- **<CONTAINER_NAME>** — один или несколько контейнеров (в том числе init-контейнеры) оригинального приложения пользователя, команда запуска которых изменена компонентом webhook на запуск файл-инжектора.

Файл-инжектор авторизуется в хранилище секретов, получает данные, передаёт секреты приложению через переменные окружения и запускает исходную команду контейнера. Если указана аннотация `secrets-store.deckhouse.io/restart-on-secret-change` со значением `watch-for-lease` или `watch-for-data`, файл-инжектор выполняет ротацию секретов при их изменении в хранилище.

Взаимодействия модуля

Модуль `secrets-store-integration` взаимодействует со следующими компонентами:

1. Kube-apiserver:

- авторизация запросов;
- получение CSIDriver, Secret, ConfigMap и ServiceAccount;
- работа с кастомными ресурсами `SecretsStoreImport`, `SecretProviderClass` и `SecretProviderClassPodStatus`.

2. Хранилище образов — получение команды запуска из образа пользовательского контейнера.

3. Хранилище секретов — авторизация запросов и получение секретов.

С модулем взаимодействуют следующие внешние компоненты:

1. Kube-apiserver — вызов mutating-вебхука при создании ресурса Pod.

2. Kubelet:

- регистрирует Node Plugin;
- проверяет `livenessProbe` CSI-драйвера;
- вызывает RPC `NodePublishVolume` и `NodeUnpublishVolume` в Node Plugin.

3. Prometheus-main — собирает метрики компонентов `csi-secrets-store` и `webhook`.

Дополнительные ресурсы

- Документация модуля `secrets-store-integration`

Подсистема Network

В данном подразделе описана архитектура подсистемы Network (сетевой подсистемы) Deckhouse Kubernetes Platform (DKP).

В подсистему Network входят следующие модули:

- **kube-dns** — устанавливает компоненты CoreDNS для управления DNS в кластере Kubernetes;

- `node-local-dns` — разворачивает кеширующий DNS-сервер на каждом узле кластера и экспортирует данные в Prometheus для анализа работы DNS в кластере на дашборде Grafana. Архитектура кеширующего DNS-сервера описана на соответствующей странице данного подраздела;
- `kube-proxy` — управляет компонентами kube-proxy для сетевого взаимодействия и балансировки нагрузки в кластере;
- `cni-cilium` — обеспечивает работу сети в кластере Kubernetes с помощью CNI Cilium;
- `cilium-hubble` — обеспечивает визуализацию сетевого стека кластера, если включен Cilium CNI;
- `ingress-nginx` — устанавливает и управляет Ingress NGINX Controller с помощью кастомных ресурсов. Архитектура модуля описана на соответствующей странице данного подраздела.
- `metallb` — реализует механизм LoadBalancer для сервисов в bare-metal-кластерах.

Также в подразделе описаны:

- архитектура кластера с включенным Istio;
- архитектура прикладного сервиса с включенным Istio.

Модуль cni-cilium

Модуль `cni-cilium` обеспечивает работу сети в кластере. Основан на проекте Cilium.

Подробнее с описанием настройки и примерами использования модуля можно ознакомиться в разделе документации модуля.

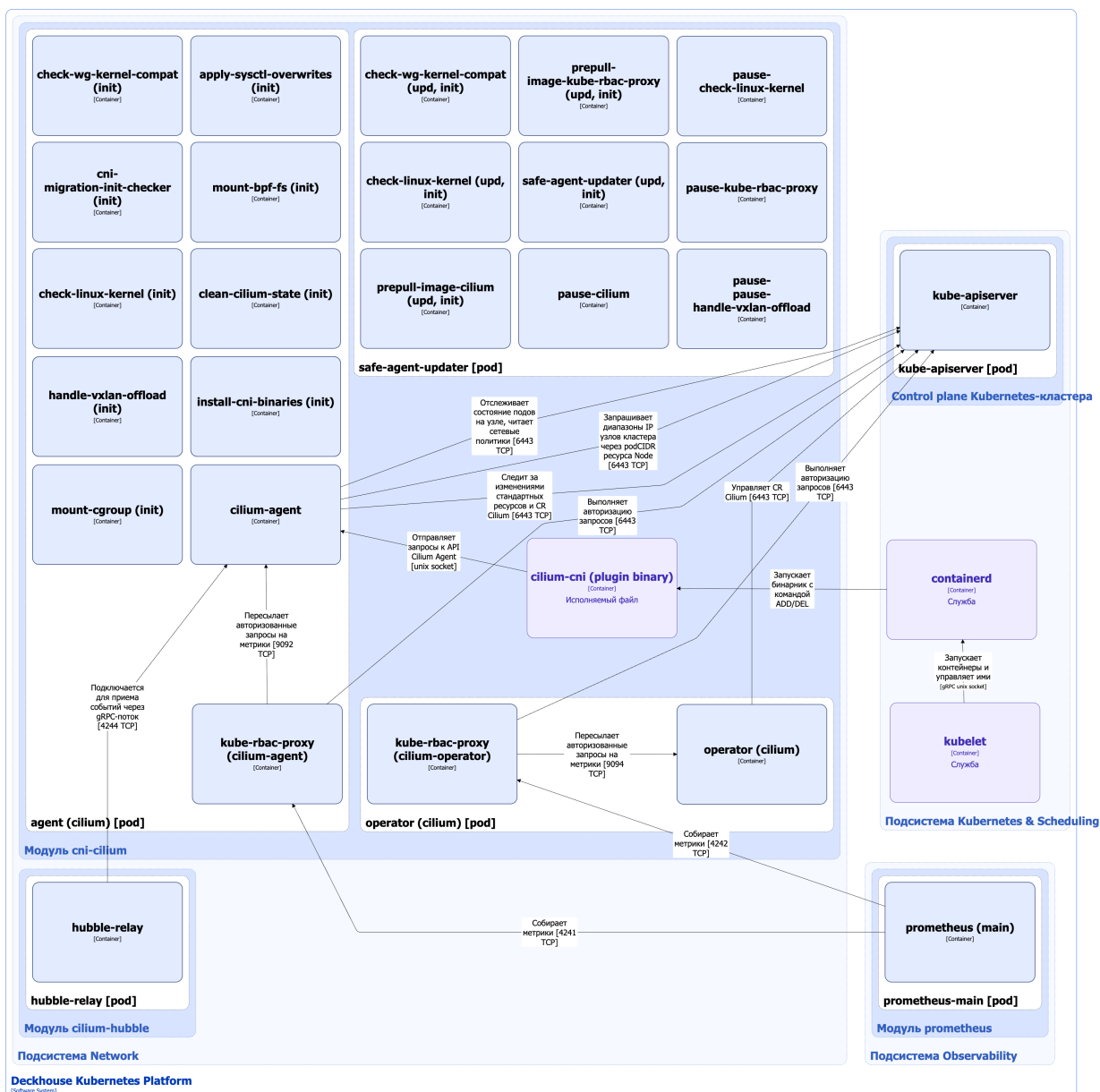
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cni-cilium` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображена на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Operator** (Deployment) — компонент в архитектуре Cilium, который берёт на себя централизованное управление глобальными задачами в кластере Kubernetes. Operator выполняет следующие функции:
 - **Управление кастомными ресурсами** — автоматически регистрирует Custom Resource Definitions (CRD), необходимые для работы Cilium, в Kubernetes API, например, CiliumEndpoint, CiliumEndpointSlice, CiliumNetworkPolicy. С полным списком CRD можно ознакомиться в документации Cilium;
 - **Сборка мусора для идентификаторов безопасности** — поддерживает локальный кеш активных идентификаторов (CiliumIdentity) и периодически сканирует его, удаляя записи об идентификаторах, которые перестали подавать сигнал активности (heartbeat). Это важно, поскольку идентификаторы представлены 16-битным целым числом, и их исчерпание (максимум 65536) может привести к проблемам;
 - **Управление CiliumEndpointSlices (CES)** — поддерживает актуальную информацию о конечных точках (endpoints) в кластере, что критично для корректной работы сетевых политик и балансировки нагрузки.

Состоит из следующих контейнеров:

- **operator** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера operator. Является Open Source-проектом.

2. **Agent** (DaemonSet) — это агент, работающий на каждом узле Kubernetes-кластера, основной компонент управления в архитектуре Cilium. Он отвечает за перевод высокоуровневых конфигураций из Kubernetes API в низкоуровневые настройки для сетевых интерфейсов и eBPF-программ, которые реально обрабатывают трафик. Agent выполняет следующие функции:
- **Управление конечными точками (Endpoints)** — отслеживает состояние всех подов (endpoints) на своём узле. Каждому поду присваивается уникальный IP-адрес и security identity (числовой идентификатор), который формируется на основе лейблов Kubernetes;
 - **Применение сетевых политик (Policy Enforcement)** — вычисляет и применяет правила сетевых политик, заданных в кластере. Cilium использует концепцию identity-based политик: правила строятся не на IP-адресах, а на security identity пода. Агент преобразует эти правила в структуры данных (BPF-карты), которые ядро Linux использует для фильтрации трафика;
 - **Управление IP-адресами (IPAM)** — выделяет IP-адреса для подов на своём узле, используя диапазоны IP-адресов из полей `spec.podCIDRs` или `spec.podCIDR` спецификации соответствующего ресурса Node;
 - **Оркестрация плоскости данных (Datapath Orchestration)** — компилирует, загружает в ядро Linux и привязывает к сетевым интерфейсам eBPF-программы. Именно эти программы в ядре и выполняют реальную обработку пакетов: пересылку, фильтрацию по политикам, отслеживание соединений, NAT и балансировку нагрузки;
 - **Реализация балансировки для сервисов (Service Load Balancing)** — настраивает правила балансировки нагрузки для сервисов в кластере, используя возможности eBPF;
 - **Обработка событий кластера.** Агент работает по событийной модели: он постоянно наблюдает за изменениями в Kubernetes API (создание подов, обновление сервисов, изменение Endpoints, обновление кастомных ресурсов) и на лету обновляет конфигурацию eBPF-программ;
 - **Обеспечение наблюдаемости (Observability).** Агент запускает локальный сервер Hubble (встроен в процесс агента), который собирает метрики и данные о сетевых потоках. Это позволяет мониторить состояние сети, выявлять проблемы и анализировать трафик между подами. Сервер Hubble запускается при включении модуля `cilium-hubble`. Компонент `hubble-relay` модуля `cilium-hubble` подключается к агентам Cilium для приема событий через gRPC-поток.

Принцип работы: когда в кластере происходит событие (например, стартует новый под), агент получает об этом уведомление и на основе текущего состояния кластера и политик настраивает eBPF-программы так, чтобы трафик для этого пода проходил корректно.

Состоит из следующих контейнеров:

- **cni-migration-init-checker** — init-контейнер, ожидающий завершения процесса миграции с другого CNI-плагины, если такая миграция была запущена;
 - **check-wg-kernel-compat** — init-контейнер, проверяющий версию ядра Linux на соответствие минимальным требованиям для работы с WireGuard, используемым в CNI Cilium;
 - **check-linux-kernel** — init-контейнер, проверяющий версию ядра Linux на соответствие минимальным требованиям для работы с CNI Cilium;
 - **clearing-unnecessary-iptables** — init-контейнер, запускающий скрипт очистки ненужных правил `iptables`;
 - **handle-vxlan-offload** — init-контейнер, исправляющий настройки UDP-сегментации (если режим работы туннеля настроен на `VXLAN`);
 - **config** — init-контейнер, генерирующий конфигурационный файл, используемый для настройки агента;
 - **mount-cgroup** — init-контейнер, настраивающий Linux cgroups (контрольные группы);
 - **apply-sysctl-overwrites** — init-контейнер, настраивающий параметры ядра Linux;
 - **mount-bpf-fs** — init-контейнер, монтирующий файловую систему BPF;
 - **clean-cilium-state** — init-контейнер, запускающий скрипт очистки состояния CNI Cilium;
 - **install-cni-binaries** — init-контейнер, запускающий скрипт установки бинарных файлов, используемых агентом;
 - **agent** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам контейнера agent. Подробно описан выше.
3. **Cilium-cni** — исполняемый файл, запускаемый компонентом `containerd`, который передает в качестве аргумента определенные команды в соответствии со спецификацией CNI, например `ADD` при запуске контейнера и `DEL` при его удалении. Cilium-cni взаимодействует

с API агента Cilium через Unix-сокеты и иницирует настройку datapath, чтобы обеспечить подключение к сети, балансировку нагрузки и сетевые политики для пода. Datapath в Cilium — это компонент, который работает в ядре Linux и который отвечает за реальную, низкоуровневую обработку сетевых пакетов в кластере Kubernetes. Проще говоря, это «дорога», по которой данные путешествуют: как пакеты идут от одного пода к другому, как маршрутизируются, как применяются сетевые политики и балансировка нагрузки.

4. **Safe-agent-updater** (DaemonSet) — специальное приложение, предназначенное для исключения сбоев, которые могут возникнуть при обновлении версии агента Cilium и привести к проблемам с сетевой доступностью компонентов DKP.

Состоит из следующих контейнеров:

- **check-wg-kernel-compat** — init-контейнер, проверяющий версию ядра Linux на соответствие минимальным требованиям для работы с WireGuard, используемым в CNI Cilium;
- **check-linux-kernel** — init-контейнер, проверяющий версию ядра Linux на соответствие минимальным требованиям для работы с CNI Cilium;
- Набор init-контейнеров для предварительного скачивания новых версий образов соответствующих контейнеров компонента agent:
 - **prepull-image-cilium**;
 - **prepull-image-kube-rbac-proxy**;
- **safe-agent-updater** — init-контейнер, который сравнивает значения специальных аннотаций в манифесте DaemonSet обновленного агента Cilium со значениями соответствующих аннотаций в метаданных запущенного на узле пода агента. В частности, в аннотации `safe-agent-updater-daemonset-generation` хранится хеш-сумма образа агента. Если хеш-суммы не совпадают, `safe-agent-updater` удаляет запущенный под и ожидает, пока под с новой версией агента не перейдет в состояние `Ready`.
- Набор сайдкар-контейнеров для предварительного скачивания новых версий образов соответствующих контейнеров компонента agent. Контейнеры стоят на паузе и выполняют только функцию хранения образов:
 - **pause-cilium**;
 - **pause-check-linux-kernel**;
 - **pause-kube-rbac-proxy**;
 - **pause-pause-handle-vxlan-offload**.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- Управляет кастомными ресурсами Cilium;
- Следит за изменениями стандартных ресурсов Kubernetes API и кастомных ресурсов Cilium;
- Отслеживает состояние подов на узле, читает сетевые политики;
- Запрашивает диапазоны IP узлов кластера через атрибуты `podCIDR/podCIDRs` спецификации ресурса Node;
- Выполняет авторизацию запросов на получение метрик компонентов agent и operator.

С модулем взаимодействуют следующие внешние компоненты:

1. **Hubble-relay** — подключается к агентам Cilium для приема событий через gRPC-поток.
2. **Prometheus-main** — собирает метрики компонентов agent и operator.
3. **Containerd** — запускает исполняемый файл `cilium-cni` с определенными командами в соответствии со спецификацией CNI, например ADD при запуске контейнера и DEL при его удалении.

Дополнительные ресурсы

- Документация модуля `cilium-hubble`
- Документация модуля `cni-cilium`

Модуль cilium-hubble

Модуль `cilium-hubble` обеспечивает визуализацию сетевого стека кластера, если включен Cilium CNI.

Подробнее с описанием настройки модуля можно ознакомиться в разделе документации модуля.

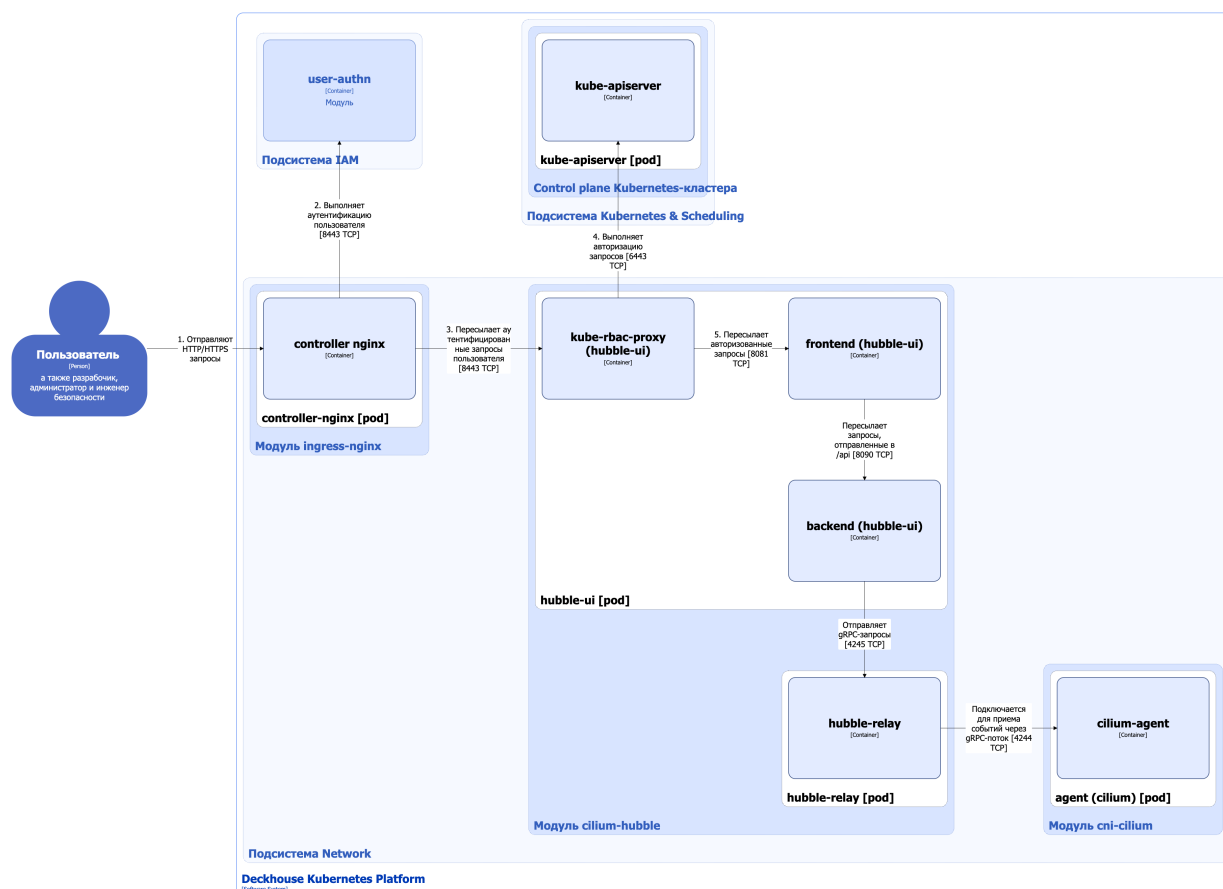
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `cilium-hubble` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображена на следующей диаграмме:



i

Номерами на схеме отмечен порядок прохождения запроса пользователя к компоненту hubble-ui:

- на шагах 1, 2 и 3 запрос проходит через Ingress NGINX Controller, где выполняется обязательная аутентификация пользователя с использованием модуля `user-authn`;
- на шагах 4 и 5 выполняется авторизация пользователя на основе Kubernetes RBAC для организации защищенного доступа.

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Hubble-relay** — компонент, который агрегирует события со всех узлов кластера в единое представление. Hubble-relay поддерживает постоянное соединение с каждым экземпляром Cilium Agent на узлах и через gRPC-поток принимает события, дедуплицирует их и предоставляет единый gRPC-эндпоинт для Hubble CLI и Hubble UI (веб-интерфейс). При этом, hubble-relay не хранит историю — он транслирует поток событий в реальном времени.

Состоит из одного контейнера:

- **hubble-relay** — основной контейнер. Является частью проекта Cilium.
2. **Hubble-ui** — компонент, реализующий веб-интерфейс. Hubble-ui подключается к hubble-relay и предоставляет удобный доступ к данным, которые поступают с агентов Cilium. Hubble-ui строит интерактивную карту сервисов (Service Map) и таблицу потоков с фильтрами.

Состоит из следующих контейнеров:

- **frontend** — контейнер, представляющий собой прокси-сервер NGINX, раздающий статические файлы веб-интерфейса hubble-ui и пересылающий запросы к эндпоинту `/api` в контейнер **backend** компонента;
- **backend** — контейнер, реализующий API hubble-ui, который отправляет запросы в компонент hubble-relay для получения данных, поступающих с агентов Cilium в реальном времени;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к веб-интерфейсу Hubble UI. Является Open Source-проектом.

Контейнеры frontend и backend собраны на основе Hubble-UI, являющегося Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Cilium-agent** — принимает поток событий с агентов Cilium в реальном времени.
2. **Kube-apiserver** — выполняет авторизацию запросов к веб-интерфейсу Hubble UI.

С модулем взаимодействуют следующие внешние компоненты:

- **Controller nginx** — пересылает внешние запросы пользователя к веб-интерфейсу модуля.

Дополнительные ресурсы

- Документация модуля `cilium-hubble`

Модуль ingress-nginx

Модуль `ingress-nginx` устанавливает и управляет Ingress NGINX Controller с помощью кастомного ресурса `IngressNginxController`.

Модуль может работать в режиме высокой доступности (HA) и предоставляет гибкие настройки размещения Ingress-контроллеров на узлах кластера, а также параметры работы контроллера с учетом особенностей реализации инфраструктуры.

Модуль поддерживает запуск и отдельную конфигурацию нескольких экземпляров Ingress NGINX Controller. Это позволяет, например, разделять внешние и внутренние (intranet) Ingress-ресурсы приложений.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

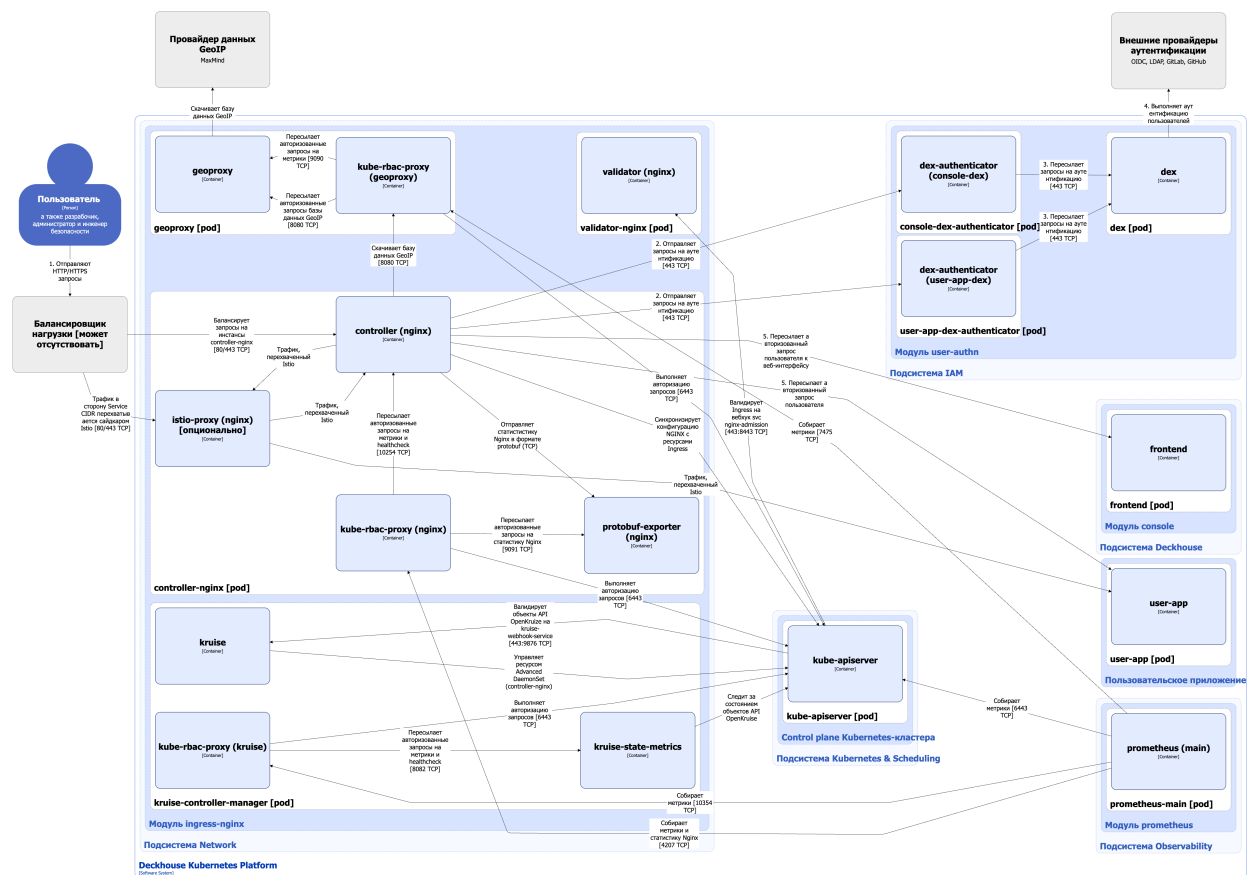
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `ingress-nginx` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller-nginx** (Advanced DaemonSet) — нестандартный DaemonSet с продвинутыми возможностями, управляемый `kruise-controller-manager`.

Состоит из следующих контейнеров:

- **init** — init-контейнер, настраивает файловую систему для ingress-controller;
 - **controller** — основной контейнер IngressNGINX controller, реализующий основную логику модуля. Является Open Source-проектом;
 - **protobuf-exporter** — сайдкар-контейнер в поде ingress-controller, принимающий статистику NGINX в виде сообщений в формате protobuf. Разбирает и агрегирует сообщения по установленным правилам, а также экспортирует метрики в формате Prometheus. Является разработкой компании «Флант»;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам и состоянию контроллера и protobuf-exporter. Является Open Source-проектом;
 - **istio-proxy** — сайдкар-контейнер Istio, добавляемый в под при включенном параметре `spec.enableIstioSidecar` кастомного ресурса IngressNginxController. В этом случае часть пользовательских запросов проходит через него.
2. **Validator-nginx** (Deployment) — состоит из одного контейнера. Validator — это Ingress NGINX Controller, запущенный в режиме валидации и обладающий ограниченным набором привилегий. Реализует вебхук-сервер, используемый для проверки Ingress-ресурсов через механику Validating Admission Controllers.
 3. **Kruise-controller-manager** (Deployment) — контроллер, управляющий кастомным ресурсом Advanced DaemonSet. Данное расширение DaemonSet позволяет использовать продвинутые возможности при обновлении Ingress NGINX controller, отсутствующие в стандартной реализации DaemonSet-контроллера Kubernetes.

Состоит из следующих контейнеров:

- **kruise** — основной контейнер kruise-controller-manager;
- **kruise-state-metrics** — сайдкар-контейнер, отслеживающий состояние объектов API OpenKruise и предоставляющий соответствующие метрики (но не метрики работы самого kruise-controller-manager);
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к метрикам и состоянию контроллера. Подробно описан выше.

4. **Geoproxy** (StatefulSet) — кеширующий прокси-сервер для Ingress-контроллера, который предоставляет быстрый доступ к базе данных GeoIP, загружаемой у провайдера MaxMind. Компонент также обеспечивает доступ к уже загруженным базам в кластерах без выхода в интернет, что повышает стабильность работы Ingress-контроллера с GeoIP-базами.

Компонент предоставляет следующие возможности:

- экономия лицензий MaxMind (загрузка баз происходит из одной точки раз в сутки);
- постоянное хранение данных (перезагрузка компонентов не приводит к повторным обращениям к серверам MaxMind);
- возможность указать своё зеркало для загрузки баз.

Состоит из следующих контейнеров:

- **geoproxy** — прокси-сервер. Является разработкой компании «Флант».
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к метрикам георгоху (сами базы GeoIP доступны без авторизации). Подробно описан выше.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - синхронизация конфигурации NGINX при изменении Ingress-ресурсов;
 - авторизация запросов на получение метрик, статистики и проверки состояния контроллера;
 - перенаправление внешних HTTP-запросов на эндпоинт API Kubernetes.
2. **Источник базы данных GeoIP** (провайдер MaxMind или зеркало) — скачивает базу данных GeoIP.
3. **Dex-authenticator служебных сервисов и пользовательских приложений** — используется для аутентификации запросов в dex через dex-authenticator, которые выполняют функции OAuth2 Proxy.
4. **Служебные сервисы DKP** (`console`, `dashboard`, Grafana и прочие) — модуль перенаправляет HTTP-запросы, прошедшие аутентификацию через Dex.
5. **Пользовательские сервисы, развернутые в DKP** — модуль перенаправляет на них внешние HTTP-запросы. Для этого пользователь должен создать соответствующие Ingress-ресурсы, а также кастомный ресурс DexAuthenticator, если требуется аутентификация через Dex.



Для упрощения схемы на ней изображены взаимодействия ingress-controller только с одним служебным сервисом DKP — компонентом frontend модуля `console` и соответствующим `console-dex-authenticator`.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — использует validation-вебхук для проверки создаваемых или обновляемых Ingress-ресурсов.
2. **Prometheus-main** — собирает метрики компонентов controller-nginx, kruise, geoproxy, а также статистику NGINX.
3. **Балансировщик нагрузки** — балансировка HTTP/HTTPS-трафика между работоспособными экземплярами ingress-controller.

Способы приема трафика из внешней сети

Способы приема трафика из внешней сети подробно описаны в параметре `spec.inlet` кастомного ресурса IngressNginxController.

Для инлетов вида `LoadBalancer`, `LoadBalancerWithProxyProtocol` и `LoadBalancerWithSSLPassthrough` указанный на схеме балансировщик нагрузки автоматически предоставляется облачным провайдером (при развертывании DKP в облаке), либо может быть реализован при помощи MetalLB-контроллера (при установке на bare-metal-хостах). С настройками модуля `metallb` можно ознакомиться в соответствующем разделе документации.

Для инлетов вида `HostPort`, `HostPortWithProxyProtocol`, `HostPortWithSSLPassthrough` и `HostWithFailover` балансировщик нагрузки разворачивается пользователем, либо может отсутствовать. В этом случае пользователь самостоятельно настраивает бэкенды балансировщика или обеспечивает сетевую связность до ingress-controller. Точкой входа в

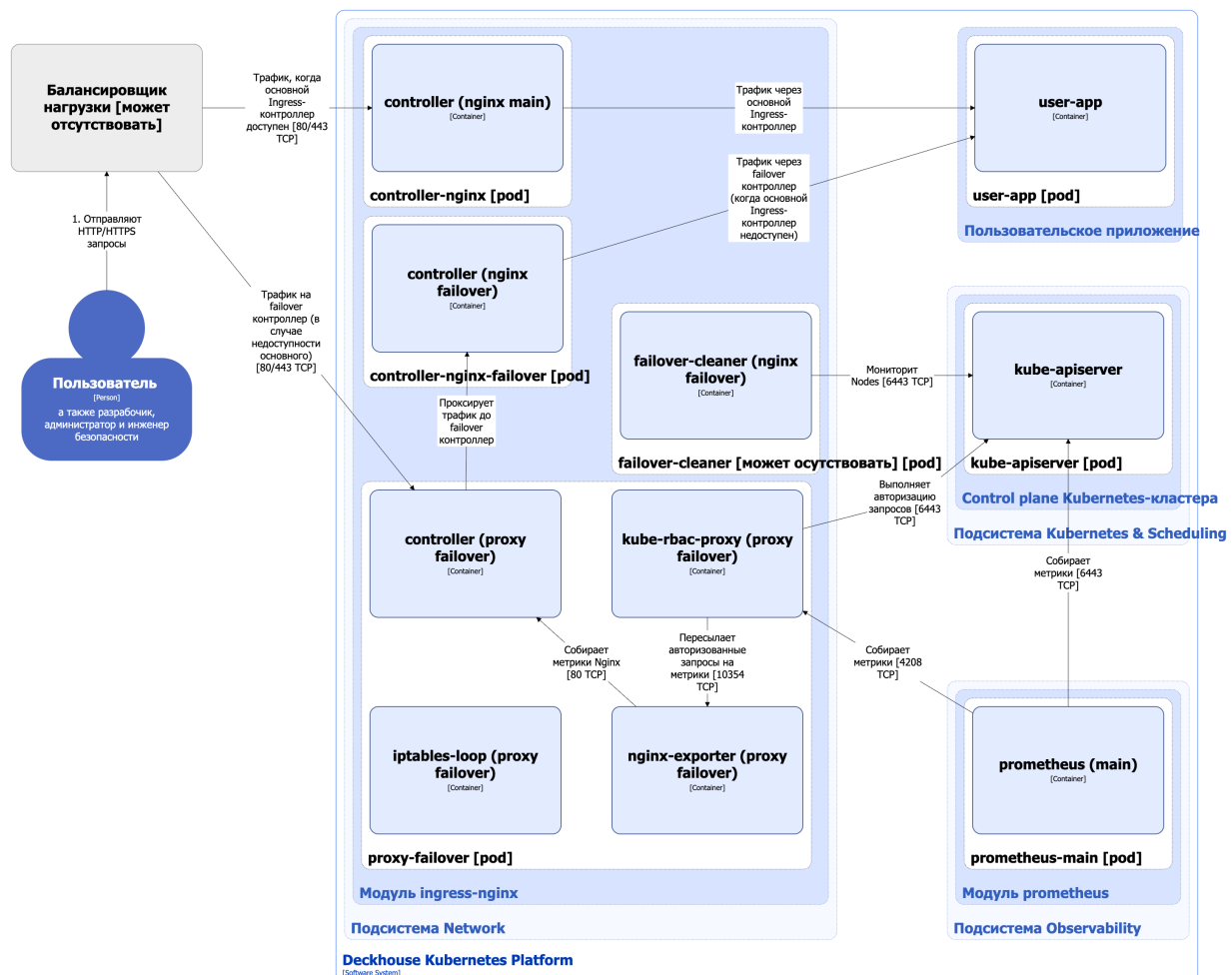
ingress-controller в этом случае являются порты на узлах кластера, на которых запущен контроллер.

Архитектура Ingress-контроллера с инлетом HostWithFailover

При значении `HostWithFailover` параметра `spec.inlet` кастомного ресурса `IngressNginxController` в кластере устанавливаются два Ingress-контроллера — основной и резервный (failover), а также proxy-failover-контроллер, который координирует переключение трафика между ними.

Основной контроллер запускается в `hostNetwork`, в то время как failover-контроллер запускается в `podNetwork`. Если под основного контроллера становится недоступен на узле, proxy-failover начинает проксировать трафик в под failover-контроллера, используя `PROXY` `PROTOCOL` для сохранения информации об IP-адресе клиента.

i На следующей схеме не показана архитектура основного Ingress-контроллера, а также взаимодействия модуля, поскольку они подробно описаны на схеме выше.



Компоненты failover Ingress-контроллера

1. **Controller-nginx-failover** (Advanced DaemonSet) — failover Ingress-контроллер, размещаемый на тех же узлах, что и основной. Состав и назначение контейнеров в поде аналогичны основному Ingress-контроллеру.
2. **Proxy-failover** (Advanced DaemonSet) — прокси-сервер.

Состоит из следующих контейнеров:

- **controller** — сервер, который выполняет следующие функции:
 - следит за работоспособностью основного Ingress-контроллера и возвращает его статус;
 - следит за файлом конфигурации NGINX на узле и перезагружает контроллер при изменении конфигурации;

- запускает экземпляр NGINX, через который трафик проксируется на failover-контроллер в случае недоступности основного. Переключение трафика на failover-контроллер и обратно на основной осуществляется с помощью правил `iptables`, которые настраивает контейнер `iptables-loop` в зависимости от доступности TCP-портов 80 и 443 основного контроллера.

Является разработкой компании «Флант»;

- **iptables-loop** — сайдкар-контейнер, который обновляет на узле правила `iptables`, необходимые для работы Ingress-контроллера. Является разработкой компании «Флант»;
 - **nginx-exporter** — сайдкар-контейнер в поде `ingress-controller`, который подключается к NGINX по протоколу HTTP и экспортирует метрики в формате Prometheus. Является Open Source-проектом;
 - **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к метрикам контроллера. Подробно описан выше.
3. **Failover-cleaner** (DaemonSet) — компонент, который разворачивается на узлах кластера, отмеченных лейблом `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true`, и выполняет очистку правил `iptables`. При штатной работе Ingress-контроллера компонент `failover-cleaner` не запущен ни на одном узле.

Взаимодействия failover Ingress-контроллера

Failover Ingress-контроллер взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов Node на предмет наличия лейбла `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true`;
 - авторизация запросов на получение метрик контроллера;
 - перенаправление внешних HTTP-запросов на эндпоинт API Kubernetes.
2. **Источник базы данных GeoIP** (провайдер MaxMind или зеркало) — скачивает базу данных GeoIP.
3. **Dex-authenticator служебных сервисов и пользовательских приложений** — используется для аутентификации запросов в dex через `dex-authenticator`, которые выполняют функции OAuth2 Proxy.
4. **Служебные сервисы DKP** (`console`, `dashboard`, Grafana и прочие) — модуль перенаправляет HTTP-запросы, прошедшие аутентификацию через Dex.
5. **Пользовательские сервисы, развернутые в DKP** — модуль перенаправляет на них внешние HTTP-запросы. Для этого пользователь должен создать соответствующие Ingress-ресурсы, а также кастомный ресурс `DexAuthenticator`, если требуется аутентификация через Dex.

С failover Ingress-контроллером взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик контроллера.
2. **Балансировщик нагрузки** — балансировка HTTP/HTTPS-трафика в случае, если основной Ingress-контроллер недоступен.

Дополнительные ресурсы

- Документация модуля `ingress-nginx`
- Параметры модуля `metallb`
- Кастомные ресурсы модуля `user-authn`

Модуль metallb

Модуль `metallb` реализует механизм LoadBalancer для сервисов в bare-metal-кластерах.

Поддерживаются следующие режимы работы:

- **Layer 2** — реализует улучшенный (по сравнению со стандартным режимом L2 в MetalLB) механизм балансировки в bare-metal-кластерах. Позволяет использовать несколько «публичных» IP-адресов для сервисов, равномерно распределяя их по доступным узлам кластера.
- **BGP** — полностью основан на решении MetalLB. Использует протокол BGP для построения маршрутов.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

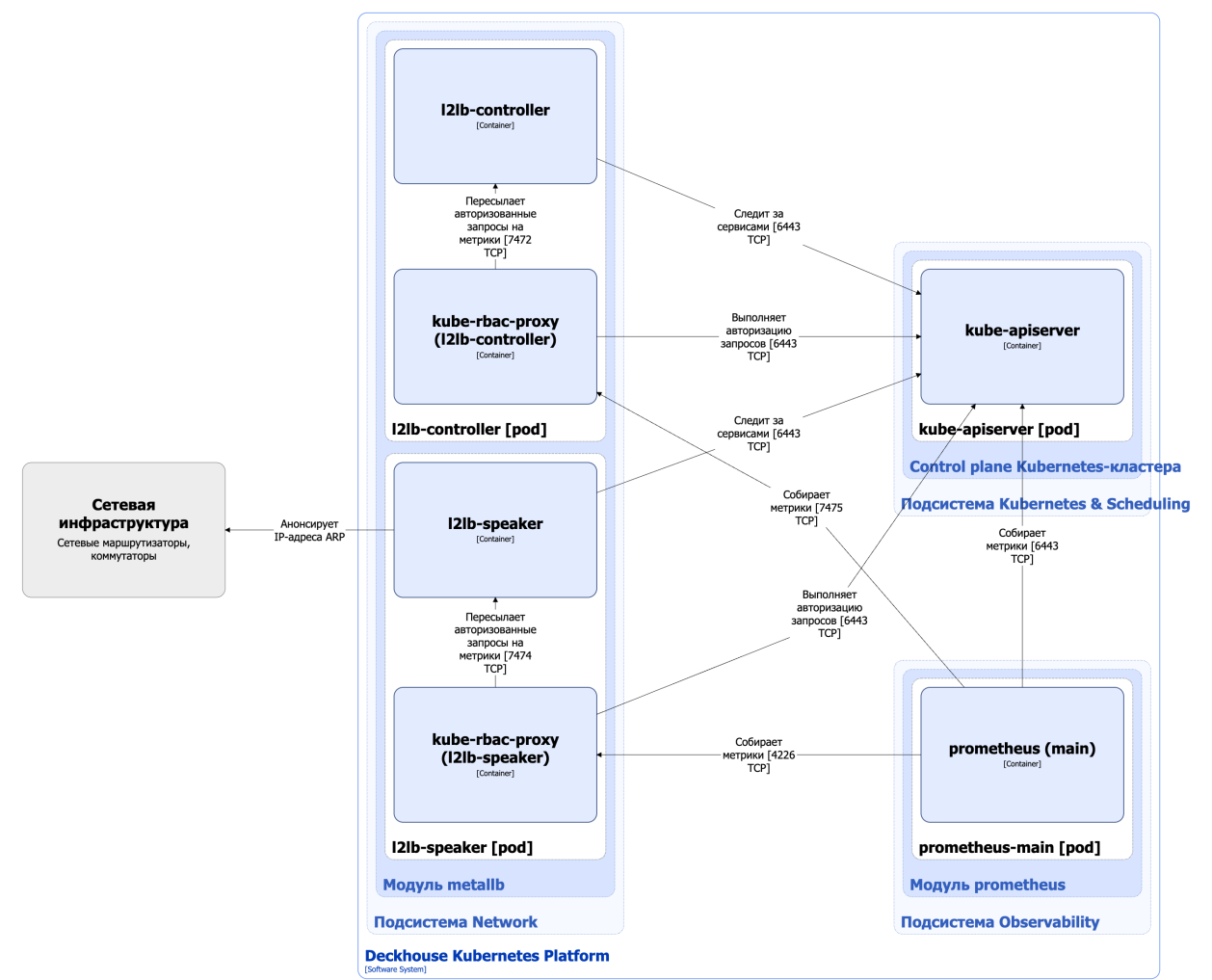
i

Для упрощения схемы приняты следующие допущения:

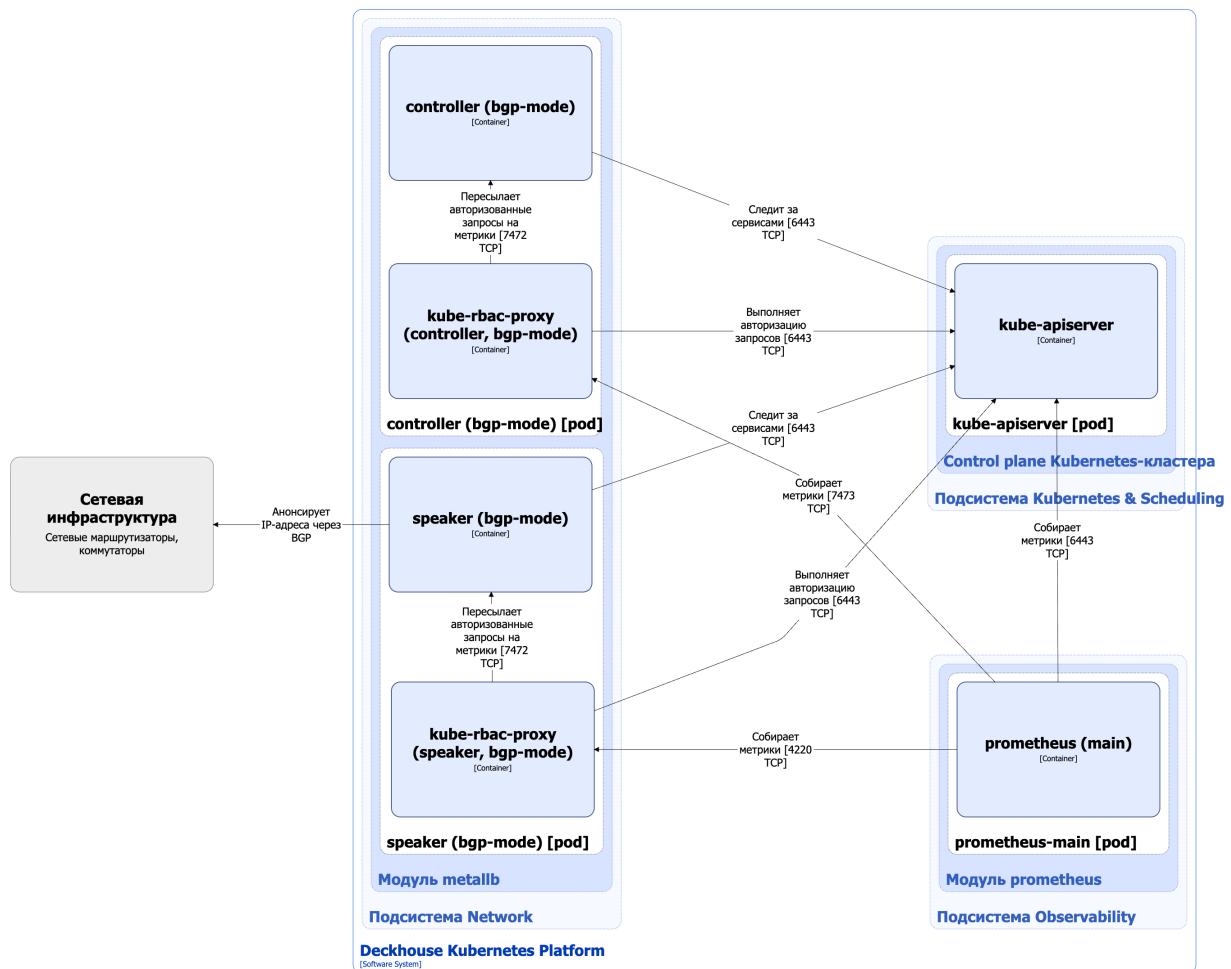
- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `metallb` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующих диаграммах.

MetallLB в режиме Layer 2:



MetallLB в режиме BGP:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller/I2lb-controller** (Deployment) — контроллер MetalLB, отвечающий за назначение IP-адресов сервисам типа LoadBalancer.

Контроллер отслеживает изменения в сервисах Kubernetes и применяет конфигурацию IP-адресов, основываясь на заранее определённом пуле адресов, указанных в настройках.

Состоит из следующих контейнеров:

- **controller/I2lb-controller** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера. Является Open Source-проектом.

2. **Speaker/I2lb-speaker** (DaemonSet) — спикер MetalLB, запускаемый на каждом узле кластера, входящем в группу балансировки. Отвечает за реализацию протокола балансировки нагрузки на уровне сети.

В зависимости от режима работы выполняет следующие функции:

- в режиме L2 использует ARP для сопоставления виртуальных IP-адресов сервисов с физическими адресами узлов;
- в режиме BGP анонсирует маршруты к виртуальным IP-адресам через протокол BGP, обеспечивая доступность сервисов за пределами кластера.

Состоит из следующих контейнеров:

- **speaker/I2lb-speaker** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам спикера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- отслеживает изменения в сервисах Kubernetes и применяет конфигурацию IP-адресов;
- авторизует запросы на получение метрик.

2. Сетевое оборудование — обеспечивает доступность виртуальных IP-адресов за пределами кластера:

- в режиме BGP — анонсирует маршруты к виртуальным IP-адресам по протоколу BGP;
- в режиме ARP — используя ARP/GARP, оповещает вышестоящий роутер о том, что виртуальные IP-адреса находятся за MAC-адресами определенных узлов.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик контроллера и спикера MetalLB.

Дополнительные ресурсы

- Документация модуля `metallb`

Модули управления DNS

Модуль `kube-dns` обеспечивает работу сервиса разрешения доменных имён на базе CoreDNS в Deckhouse Kubernetes Platform (DKP).

Подробнее о настройках модуля и примерах его использования можно узнать в соответствующем разделе документации.

Модуль kube-dns

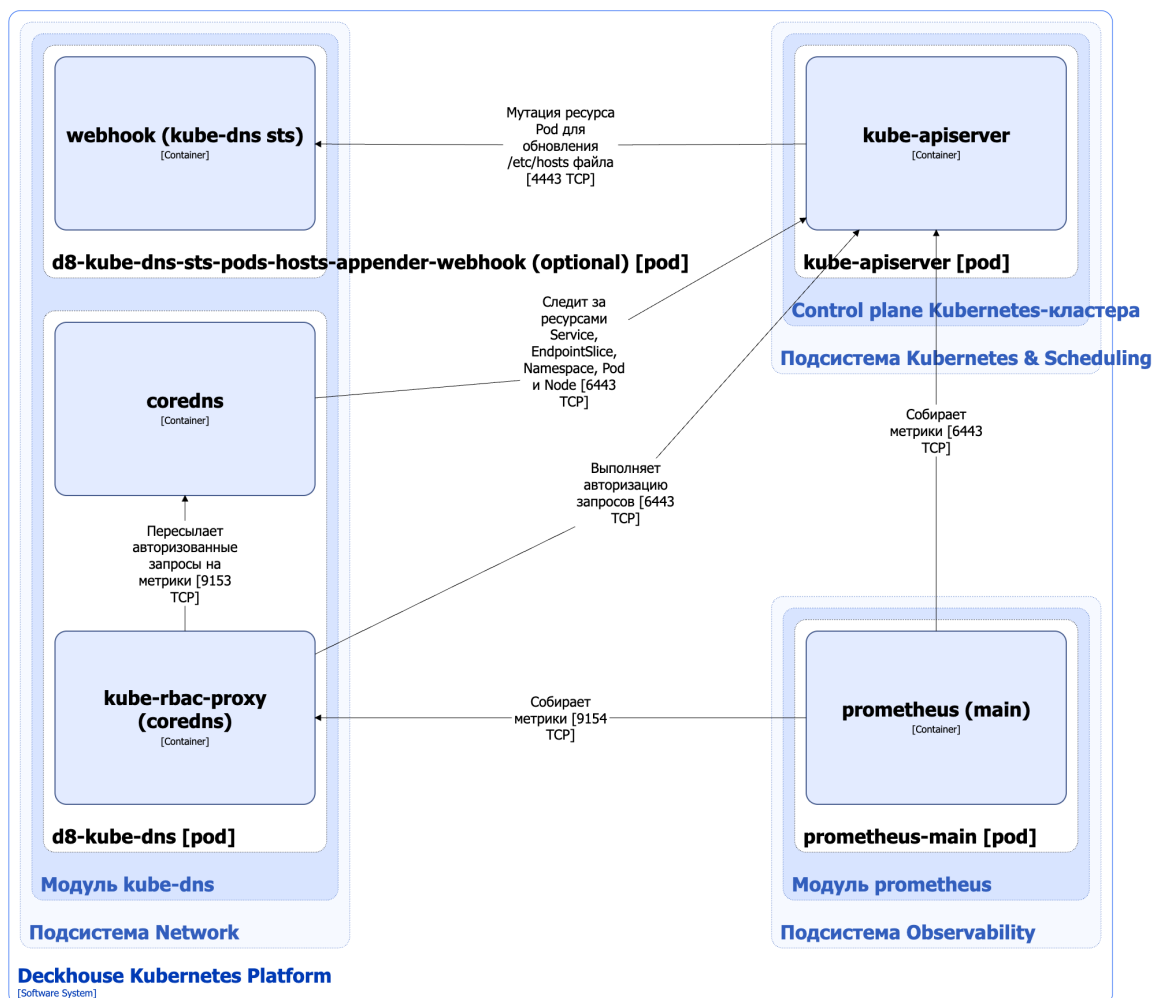
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `kube-dns` на уровне 2 модели C4 и его взаимодействие с другими компонентами Deckhouse Kubernetes Platform (DKP) показаны на следующей диаграмме:



Компоненты модуля

Модуль `kube-dns` состоит из следующих компонентов:

1. **D8-kube-dns** (Deployment) — основной компонент модуля, реализующий DNS-сервер в кластере Kubernetes.

Компонент `d8-kube-dns` отслеживает изменения стандартных ресурсов `Service`, `EndpointSlice`, `Namespace`, `Pod`, а также периодически запрашивает ресурсы `Node`. На основе полученных данных `d8-kube-dns` обновляет записи в локальной базе объектов.

Состоит из следующих контейнеров:

- **coredns** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищённого доступа к метрикам `coredns`. Является Open Source-проектом.

2. **D8-kube-dns-sts-pods-hosts-appender-webhook** (Deployment) — опциональный компонент, состоящий из одного контейнера **webhook**.

Deckhouse-контроллер модуля `deckhouse` создаёт этот компонент, если в `ModuleConfig` задан параметр `.spec.settings.clusterDomainAliases`.

Компонент реализует мутирующий вебхук-сервер, добавляющий init-контейнер **render-etc-hosts-with-cluster-domain-aliases** в под, созданный StatefulSet-контроллером, если в спецификации пода указана опция `.spec.subdomain`.

Init-контейнер изменяет файл `/etc/hosts`, чтобы подсистема разрешения имён корректно работала с алиасами домена кластера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

- **Kube-apiserver:**

- наблюдение за стандартными ресурсами Service, Endpoint, EndpointSlice, Namespace, Pod и Node;
- авторизация запросов на получение метрик.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — изменение ресурсов Pod, созданных StatefulSet-контроллером.
2. **Prometheus-main** — собирает метрики модуля.

Модуль node-local-dns

Модуль `node-local-dns` предоставляет на каждом узле кластера кеширующий DNS-сервис и механизм перенаправления DNS-запросов, тем самым снижая нагрузку на `coredns` модуля `kube-dns`.

В зависимости от используемого CNI-плагина модуль `node-local-dns` реализует разные механизмы перенаправления DNS-запросов:

- при использовании `cni-cilium` за перенаправление DNS-запросов отвечает Cilium, который использует для этого кастомный ресурс `CiliumLocalRedirectPolicy`;
- при использовании `cni-flannel` или `cni-simple-bridge` за перенаправление DNS-запросов отвечает `iptables`.

Подробнее о настройках модуля и примерах его использования можно узнать в соответствующем разделе документации.

При использовании cni-cilium

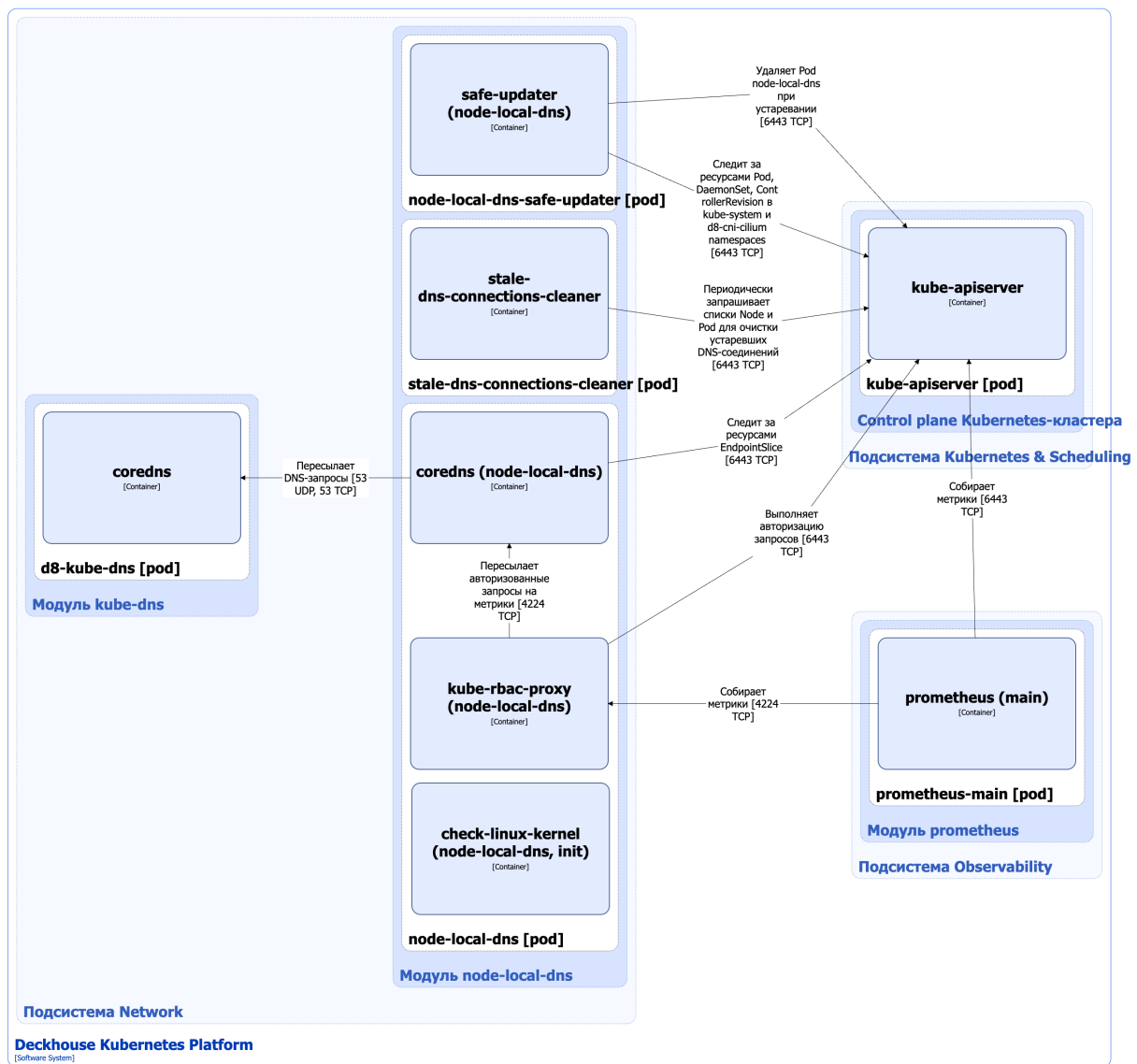
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-local-dns` при использовании Cilium в качестве CNI-плагина на уровне 2 модели C4 и его взаимодействие с другими компонентами Deckhouse Kubernetes Platform (DKP) показаны на следующей диаграмме:



Компоненты модуля

Модуль `node-local-dns` состоит из следующих компонентов:

1. **Node-local-dns** (DaemonSet) — основной компонент модуля, реализующий кеширующий DNS-сервер в кластере Kubernetes.

Компонент `node-local-dns` отслеживает изменения стандартных ресурсов `EndpointSlice` и на их основе обновляет список DNS-серверов для перенаправления запросов.

Состоит из следующих контейнеров:

- **check-linux-kernel** — init-контейнер, выполняющий проверку версии ядра Linux;
- **coredns** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для защищенного доступа к метрикам `coredns`. Является Open Source-проектом.

2. **Stale-dns-connections-cleaner** (DaemonSet) — компонент, который удаляет устаревшие UDP-соединения, оставшиеся после перезапуска Pod `node-local-dns`. Состоит из одного контейнера **stale-dns-connections-cleaner**.



Компонент имеет привилегированный доступ к сетевой системе каждого узла. В Linux для этого требуется capability `CAP_NET_ADMIN`. Такой доступ необходим для выполнения операций с сетевыми подключениями на уровне ядра ОС Linux.

3. **Safe-updater** (Deployment) — компонент, обеспечивающий безопасный перезапуск `node-local-dns` при изменении спецификации DaemonSet.

Safe-updater проверяет, что Cilium на узле запущен и находится в корректном состоянии, и

только после этого отправляет команду на удаление Pod `node-local-dns`.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- наблюдение за стандартными ресурсами EndpointSlice, DaemonSet, ControllerRevision и Pod;
- периодическое получение ресурсов Node;
- удаление Pod `node-local-dns` при устаревании конфигурации DaemonSet;
- авторизация запросов на получение метрик.

2. D8-kube-dns — выполнение DNS-запросов.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — собирает метрики модуля.

При использовании cni-flannel или cni-simple-bridge

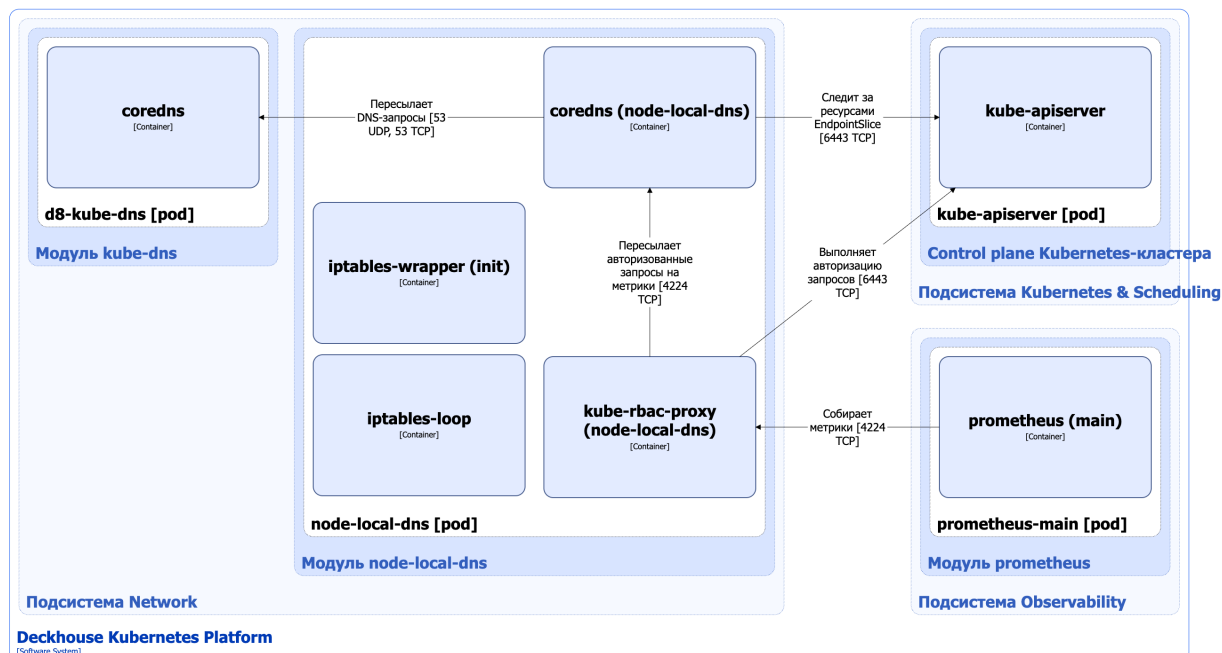
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `node-local-dns` при использовании CNI-плагина `cni-flannel` или `cni-simple-bridge` на уровне 2 модели C4 и его взаимодействие с другими компонентами Deckhouse Kubernetes Platform (DKP) показаны на следующей диаграмме:



Компоненты модуля

Модуль `node-local-dns` состоит из следующих компонентов:

- **Node-local-dns** (DaemonSet) — основной компонент модуля, реализующий кеширующий DNS-сервер в кластере Kubernetes.

Компонент `node-local-dns` отслеживает изменения стандартных ресурсов `EndpointSlice` и на их основе обновляет список DNS-серверов для перенаправления запросов.

Состоит из следующих контейнеров:

- **iptables-wrapper** — `init`-контейнер, выполняющий подготовку необходимых для работы с `iptables` исполняемых файлов;
- **coredns** — основной контейнер;
- **iptables-loop** — сайдкар-контейнер, обновляющий `iptables`-правила перенаправления DNS-запросов с учётом готовности `node-local-dns`;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищённого доступа к метрикам контейнера `coredns`. Является Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- наблюдение за стандартными ресурсами `EndpointSlice`;
- авторизация запросов на получение метрик.

2. D8-kube-dns — выполнение DNS-запросов.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — собирает метрики модуля.

Дополнительные ресурсы

- Параметры модуля `kube-dns`
- Параметры модуля `node-local-dns`
- Кеширующий DNS-сервер в кластере
- Документация модуля `deckhouse`
- Документация модуля `kube-dns`

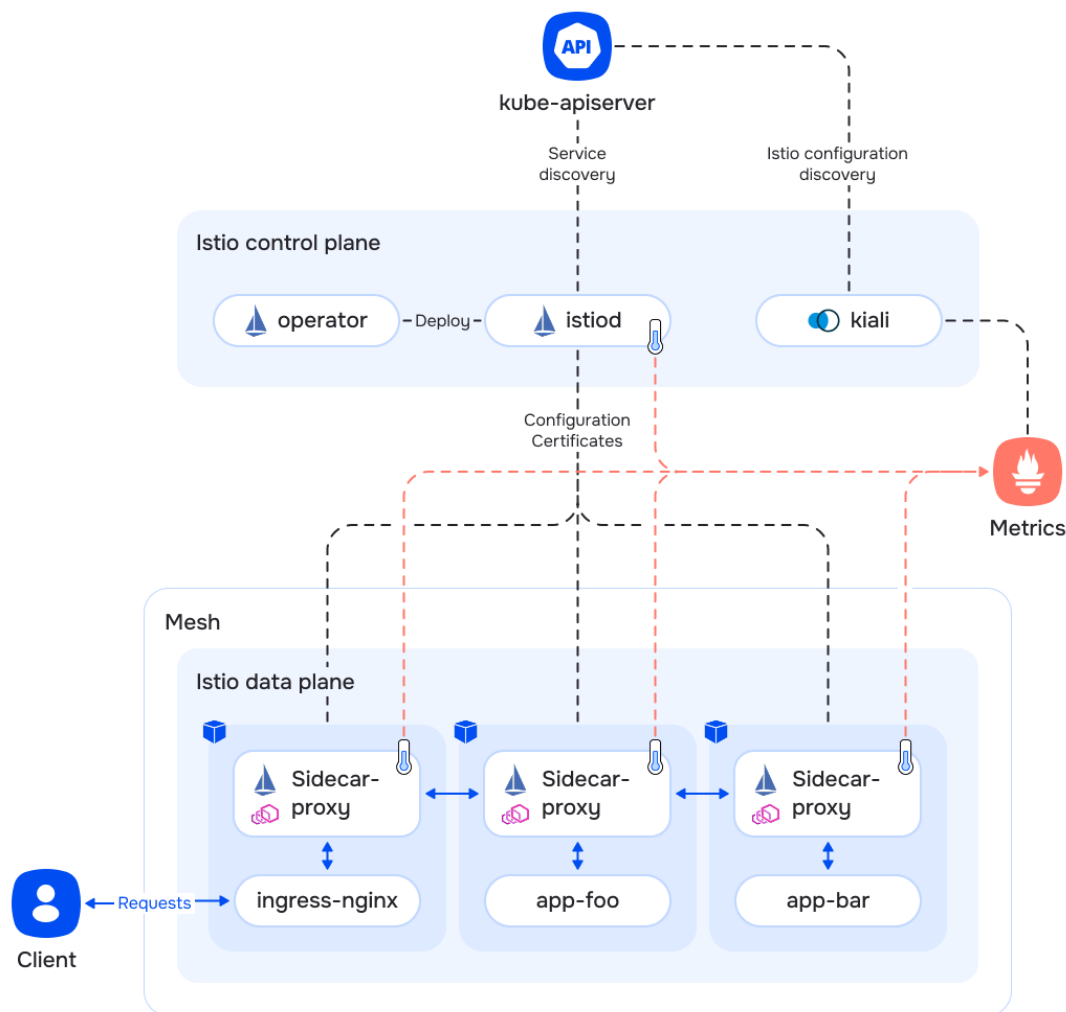
Архитектура Istio на уровне кластера

i

Совместимость версий Istio с версиями Kubernetes определяется таблицей совместимости в документации модуля `istio`. Перед использованием ориентируйтесь на актуальные версии и их статус из этого раздела.

Компоненты Istio делятся на две категории:

- **Control plane** — управляющие и обслуживающие сервисы. Под `control plane` обычно подразумевают поды `istiod`.
- **Data plane** — прикладная часть Istio. Представляет собой контейнеры `sidecar-proxy`.



Все сервисы из data plane группируются в сервис-меш (service mesh). Его характеристики:

- Общий неймспейс для генерации идентификатора сервиса в формате `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. У каждого сервис-меш есть идентификатор `TrustDomain`, который в данном случае совпадает с доменом кластера. Например, `mycluster.local/ns/myns/sa/myapp`.
- Аутентификация сервисов внутри одного сервис-меш с помощью доверенных корневых сертификатов.

Элементы control plane:

- **Istiod** — ключевой сервис, обеспечивающий решение следующих задач:
 - Непрерывная связь с API Kubernetes и сбор информации о прикладных сервисах.
 - Обработка и валидация с помощью механизма Kubernetes Validating Webhook всех кастомных ресурсов, которые связаны с Istio.
 - Компоновка конфигурации для каждого `sidecar-proxy` индивидуально:
 - генерация правил авторизации, маршрутизации, балансировки и пр.;
 - распространение информации о других прикладных сервисах в кластере;
 - выпуск индивидуальных клиентских сертификатов для организации схемы Mutual TLS. Эти сертификаты не связаны с сертификатами, которые использует и контролирует сам Kubernetes для своих служебных нужд.
 - Автоматическая подстройка манифестов, определяющих прикладные поды через механизм Kubernetes Mutating Webhook:
 - внедрение дополнительного служебного контейнера `sidecar-proxy`;
 - внедрение дополнительного `init`-контейнера для адаптации сетевой подсистемы (настройка DNAT для перехвата прикладного трафика);
 - перенаправление `readiness`- и `liveness`-проб через `sidecar-proxy`.
- **Operator** — компонент, отвечающий за установку всех ресурсов, необходимых для работы control plane определенной версии.
- **Kiali** — панель управления и наблюдения за ресурсами Istio и пользовательскими сервисами под управлением Istio:
 - визуализация связей между сервисами;
 - диагностика проблемных связей;

- диагностика состояния control plane.

С включенным Istio поведение Ingress-контроллера изменяется следующим образом:

- К подам контроллера добавляется sidecar-проху, который обслуживает только трафик от контроллера в сторону прикладных сервисов (параметр `enableIstioSidecar` ресурса `IngressNginxController`).
- Сервисы, которые не находятся под управлением Istio, продолжают работать в прежнем режиме, без перехвата запросов сайдкарм контроллера.
- Запросы к сервисам под управлением Istio перехватываются сайдкарм и обрабатываются в соответствии с правилами Istio. Подробнее об активации Istio для приложений можно почитать в соответствующем разделе документации.

Контроллер istiod и каждый контейнер sidecar-проху экспортируют собственные метрики, которые собирает кластерный Prometheus.

Накладные расходы

Внедрение Istio повлечёт за собой дополнительные расходы ресурсов, как для control plane (контроллер istiod), так и для data plane (istio-сайдкары приложений).

Control plane

Контроллер istiod непрерывно наблюдает за конфигурацией кластера, компонует настройки для istio-сайдкаров data plane и рассылает их по сети. Соответственно, чем больше приложений и их экземпляров, чем больше сервисов и чем чаще эта конфигурация меняется, тем больше требуется вычислительных ресурсов и больше нагрузка на сеть.

Поддерживается два подхода к снижению нагрузки на экземпляры контроллеров:

- горизонтальное масштабирование (параметр `controlPlane.replicasManagement` модуля `istio`) — чем больше экземпляров контроллеров, тем меньше экземпляров istio-сайдкаров обслуживать каждому из них и тем меньше нагрузка на CPU и на сеть;
- сегментация data plane с помощью ресурса `Sidecar` (рекомендуемый подход) — чем меньше область видимости у отдельного istio-сайдкара, тем меньше требуется обновлять данных в data plane и тем меньше нагрузка на CPU и на сеть.

Примерная оценка накладных расходов для экземпляра control plane, который обслуживает 1000 сервисов и 2000 istio-сайдкаров — 1 vCPU и 1,5 ГБ RAM.

Data plane

На потребление ресурсов data plane (istio-сайдкары) влияет множество факторов:

- количество соединений;
- интенсивность запросов;
- размер запросов и ответов;
- протокол (HTTP/TCP);
- количество ядер CPU;
- сложность конфигурации сервис-меш.

Примерная оценка накладных расходов для экземпляра istio-сайдкара — 0,5 vCPU на 1000 запросов/сек и 50 МБ RAM.

Istio-сайдкары также вносят задержку в сетевые запросы — примерно 2,5 мс на запрос.

Дополнительные ресурсы

- Кастомные ресурсы модуля `ingress-nginx`
- Документация модуля `istio`

Архитектура прикладного сервиса с включенным Istio

i

Совместимость версий Istio с версиями Kubernetes определяется таблицей совместимости в документации модуля `istio`. Перед использованием ориентируйтесь на актуальные версии и их статус из этого раздела.

Особенности архитектуры

- **Sidecar-proxy:** Каждый под прикладного сервиса получает дополнительный контейнер —

- sidecar-proxy, содержащий два приложения:
 - **Envoy** — проксирует прикладной трафик и реализует все функции, которые предоставляет Istio, включая маршрутизацию, аутентификацию, авторизацию и пр.
 - **Pilot-agent** — часть Istio, отвечающая за поддержание конфигурации Envoy в актуальном состоянии. Включает в себя кеширующий DNS-сервер.
- **Настройки DNAT:**
 - В каждом поде с помощью дополнительного init-контейнера настраивается DNAT входящих и исходящих прикладных запросов в sidecar-proxy. В результате трафик будет перехватываться прозрачно для приложений.
 - Поскольку входящий трафик перенаправляется в sidecar-proxy, это касается и readiness/liveness-проб. Так как подсистема Kubernetes не поддерживает пробы в формате Mutual TLS, все существующие пробы перенастраиваются на порт в sidecar-proxy, который передает их приложению без изменений.
- **Ingress-контроллер:**
 - Каждый под Ingress-контроллера также включает sidecar-proxy, который обрабатывает трафик между контроллером и сервисами.
 - Входящий трафик от пользователей обрабатывается непосредственно контроллером.
- **Ресурсы типа Ingress:** Эти ресурсы требуют минимальной доработки в виде добавления аннотаций:
 - `nginx.ingress.kubernetes.io/service-upstream: "true"` — Ingress-контроллер в качестве upstream будет использовать ClusterIP сервиса вместо адресов подов. Балансировкой трафика между подами теперь занимается sidecar-proxy. Используйте эту опцию только если у вашего сервиса есть ClusterIP.
 - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — sidecar-proxy Ingress-контроллера принимает решения о маршрутизации на основе заголовка Host. Без данной аннотации контроллер оставит заголовок с адресом сайта, например Host: example.com.
- **Сервисы:**
 - Ресурсы типа Service не требуют изменений и продолжают работать без адаптации. Приложениям все так же доступны адреса сервисов вида `servicename`, `servicename.myns.svc` и пр.
- **DNS-запросы:**
 - Внутренние DNS-запросы подов прозрачно перенаправляются на обработку в sidecar-proxy для разрешения DNS-имен сервисов из соседних кластеров.

Жизненный цикл пользовательского запроса

Приложение с выключенным Istio

Приложение с включенным Istio

Дополнительные ресурсы

- Документация модуля `istio`

Кеширующий DNS-сервер в кластере

В стандартной работе DNS в Kubernetes существует ряд проблем, которые могут привести к неоправданному снижению ключевых показателей работы сервиса:

- Linux-ядро по умолчанию не кеширует DNS-запросы, поэтому в подах локальный кеш отсутствует.
- Все DNS-запросы из контейнера влекут за собой сетевой запрос к кластерному DNS. Запросы к ресурсам в рамках одного узла все равно приводят к сетевым запросам.
- Запрос из пода сначала разрешается в кластерных DNS-зонах и только потом отправляется на внешние DNS-серверы. Например, запрос на `ya.ru` будет разрешаться сначала в кластерных зонах, таких как `cluster.local`, `svc.cluster.local`, `<namespace>.svc.cluster.local`, и только после получения отрицательных ответов, то есть по сути только с более чем второго раза, будет разрешаться правильно.

При появлении небольших сетевых задержек качество сервиса может значительно деградировать из-за приведенных выше проблем.

Одно из решений — установить DNS-сервер на каждый узел. В Deckhouse Kubernetes Platform это реализуется с помощью модуля `node-local-dns`.

При использовании кеширующего DNS-сервера внешние запросы (но только отсутствующие в кеше) также будут сначала пытаться разрешаться по цепочке внутренних зон. При большой нагрузке (при большом количестве запросов на одни и те же записи в секунду, как это часто и бывает) кеширования достаточно для значительного улучшения DNS-резолвинга.

Принцип работы кеширующего DNS-сервера

При разворачивании кеширующего DNS-сервера модуль `node-local-dns` выполняет на каждом узле кластера следующие настройки:

- Настраивает интерфейс с IP-адресом clusterIP сервиса `kube-dns`.
- Запускает кеширующий CoreDNS, который слушает на этом адресе.
- Добавляет правило в iptables: если сокет открыт, трафик перенаправляется на него, если не открыт — работает обычная Kubernetes-маршрутизация через ClusterIP:

```
-A PREROUTING -d <IP-адрес kube-dns> -m socket --nowildcard -j NOTRACK
```

Особенности конфигурации CoreDNS

Основные характеристики конфигурации CoreDNS:

- кеширование всех запросов;
- передача всех DNS-запросов в ClusterIP кластерного DNS.

Дашборд Grafana

Дашборд `Kubernetes / DNS (node local)` отображает:

- общие графики (для оценки работы DNS в целом);
- графики по узлам (для глубокого изучения найденной на общем графике проблемы узла);
- графики по upstream (для оценки работы кластерного DNS и серверов узлов, указанных в `/etc/resolv.conf`).

Дополнительные ресурсы

- Документация модуля `node-local-dns`

Подсистема Storage

В данном подразделе описана архитектура подсистемы Storage (подсистемы хранения) Deckhouse Kubernetes Platform (DKP).

В подсистему Storage входят следующие модули:

- `local-path-provisioner` — предоставляет локальное хранилище на узлах Kubernetes с использованием томов `HostPath` и создает ресурсы `StorageClass` для управления выделением локального хранилища;
- `snapshot-controller` — включает поддержку снимков для совместимых CSI-драйверов в кластере Kubernetes;
- `sds-local-volume` — управляет локальными блочными хранилищами на базе LVM и позволяет создавать ресурсы `StorageClass` в Kubernetes с помощью кастомного ресурса `LocalStorageClass`;
- `sds-node-configurator` — управляет блочными устройствами и LVM на узлах Kubernetes-кластера через кастомные ресурсы;
- `sds-replicated-volume` — управляет реплицируемым блочным хранилищем на базе DRBD. В качестве control plane используется LINSTOR;
- `storage-volume-data-manager` — обеспечивает безопасный экспорт и импорт содержимого постоянных томов по протоколу HTTP;
- модули, предоставляющие реализацию CSI-драйвера для интеграции с различными типами хранилищ (программными и аппаратными):
 - `csi-ceph`;
 - `csi-hpe`;
 - `csi-huawei`;
 - `csi-netapp`;
 - `csi-nfs`;
 - `csi-s3`;
 - `csi-scsi-generic`;
 - `csi-vsphere`;
 - `csi-yadro-tatlin-unified`.

Модуль local-path-provisioner

Модуль `local-path-provisioner` предоставляет локальное хранилище на узлах Kubernetes с

использованием томов `HostPath` и создает ресурсы `StorageClass` для управления выделением локального хранилища.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

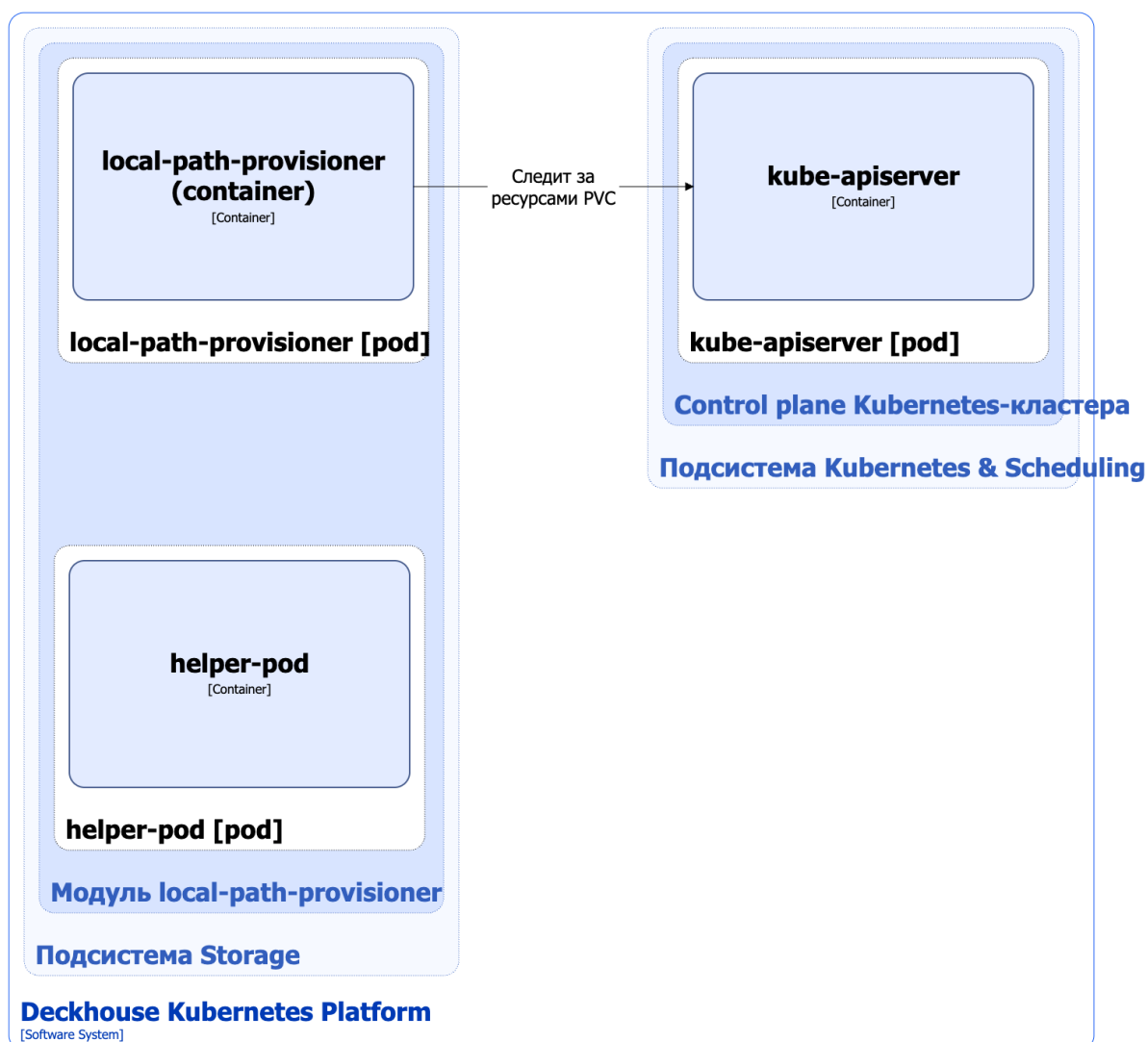
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `local-path-provisioner` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Local-path-provisioner** — выполняет следующие действия, когда под заказывает диск:
 - создает `PersistentVolume` с типом тома `HostPath`;
 - создает на нужном узле локальный каталог для тома. Путь к каталогу формируется на

основе параметра `path` кастомного ресурса `LocalPathProvisioner`, а также имени `PersistentVolume` и `PersistentVolumeClaim`.

Состоит из одного контейнера:

- **local-path-provisioner** — является Open Source-проектом.
2. **Helper-pod** — запускает на узле скрипт установки (`SETUP`) перед созданием тома, чтобы подготовить каталог тома на узле, и скрипт очистки (`TEARDOWN`) после удаления тома, чтобы очистить каталог тома на узле.

Состоит из одного контейнера:

- **helper-pod**.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - мониторинг ресурсов `PersistentVolumeClaim`.

Дополнительные ресурсы

- Документация модуля `local-path-provisioner`

Модуль snapshot-controller

Модуль `snapshot-controller` включает поддержку снимков томов для совместимых CSI-драйверов в Deckhouse Kubernetes Platform (DKP).

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

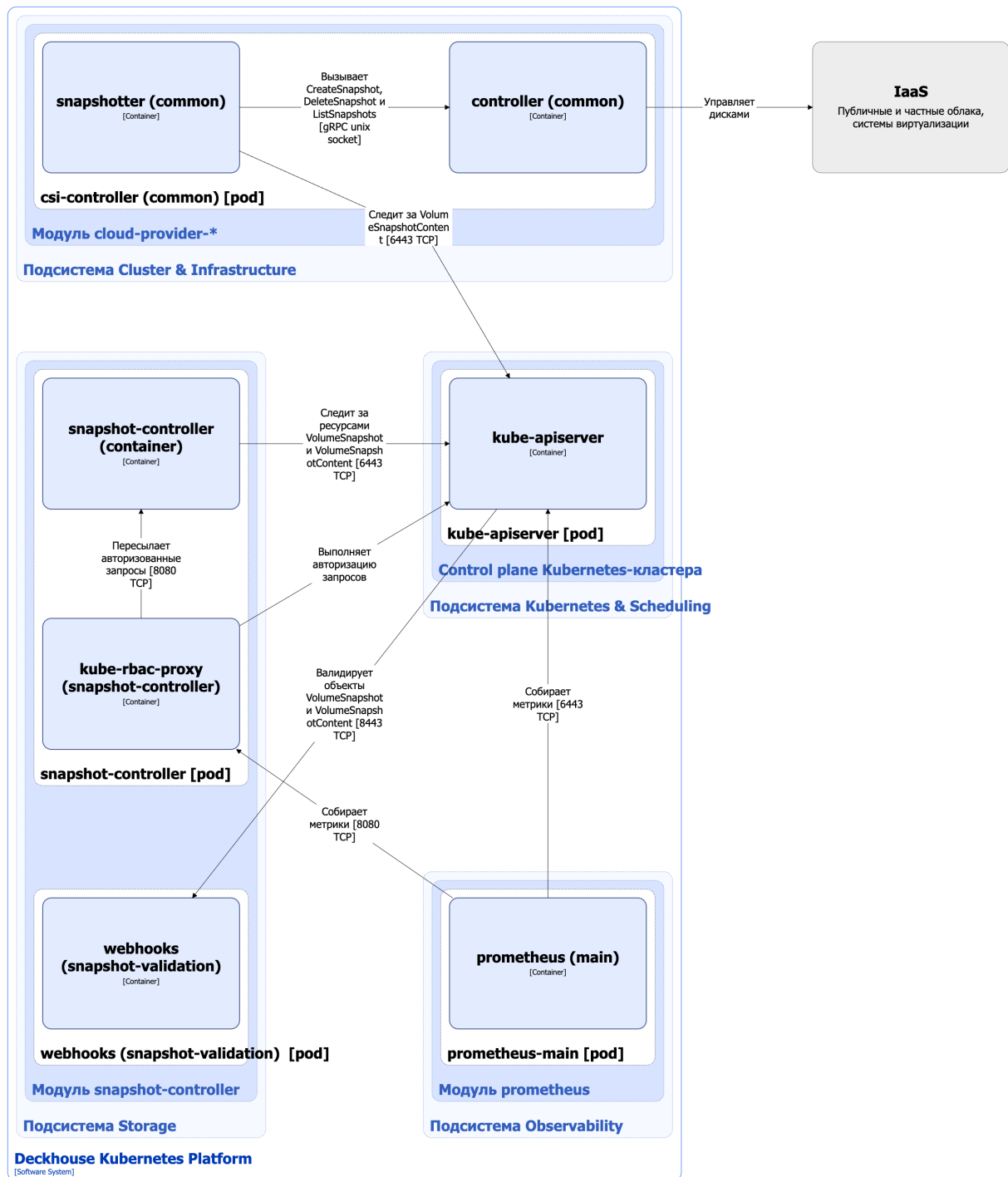
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `snapshot-controller` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Snapshot-controller** — контроллер снимков, работающий совместно с сайдкар-контейнером **snapshotter** (**external-snapshotter**) пода **csi-controller** модуля **cloud-provider-*** (при условии, что CSI-драйвер провайдера поддерживает создание снимков).

Для всех установленных CSI-драйверов используется один **snapshot-controller**, который следит за ресурсами **VolumeSnapshot** и **VolumeSnapshotContent**. При появлении нового ресурса **VolumeSnapshot** контроллер создает ресурс **VolumeSnapshotContent** и связывает их между собой. В результате **VolumeSnapshot** ссылается на соответствующий **VolumeSnapshotContent**, а тот — на исходный **VolumeSnapshot**.

Создание снимка — это многоступенчатый процесс:

1. **Snapshot-controller** создает ресурс **VolumeSnapshotContent**.
2. Сайдкар **snapshotter** запускает создание снимка с помощью **csi-controller** на соответствующем узле и обновляет статус **VolumeSnapshotContent** (поля **snapshotHandle**, **creationTime**, **restoreSize**, **readyToUse** и **error**).

3. **Snapshot-controller** отслеживает статус `VolumeSnapshotContent` и обновляет статус ресурса `VolumeSnapshot` до завершения двунаправленной привязки и установки поля `readyToUse` в значение `true`. При возникновении ошибки аналогичным образом обновляется поле `error`.

Состоит из следующих контейнеров:

- **snapshot-controller** — является Open Source-проектом;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера. Является Open Source-проектом.
2. **Webhooks** — компонент, реализующий вебхук-сервер для валидации ресурсов `VolumeSnapshot` и `VolumeSnapshotContent` с помощью механизма `Validating Admission Controllers`. Состоит из одного контейнера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов `VolumeSnapshot` и `VolumeSnapshotContent`;
- авторизация запросов на получение метрик контроллера.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — использование вебхука валидации для проверки создаваемых ресурсов `VolumeSnapshot` и `VolumeSnapshotContent`.
2. **Prometheus-main** — сбор метрик контроллера.

Дополнительные ресурсы

- Документация модуля `snapshot-controller`

Модуль sds-local-volume

Модуль `sds-local-volume` предназначен для управления локальным блочным хранилищем на базе LVM. Он позволяет создавать `StorageClass` в Kubernetes с помощью ресурса `LocalStorageClass`.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

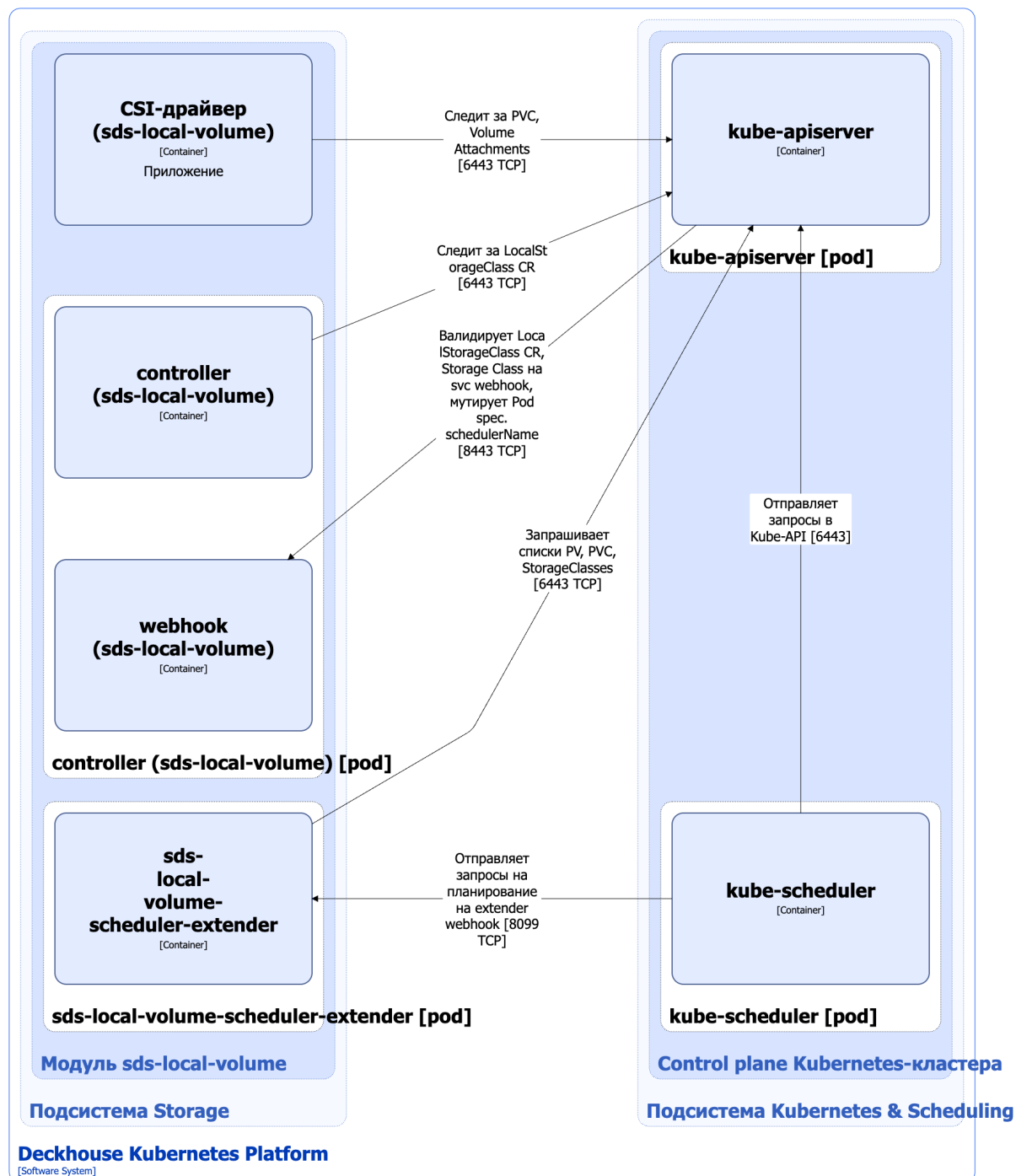
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `sds-local-volume` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий кастомные ресурсы LocalStorageClass. LocalStorageClass — пользовательский ресурс Kubernetes, определяющий конфигурацию для Kubernetes StorageClass. Создаваемый StorageClass использует `local.csi.storage.deckhouse.io` provisioner. В StorageClass задаются типы логических томов LVM, параметры VolumeGroups, reclaim policy, volume binding mode и другие настройки. Эти параметры используются provisioner'ом CSI-драйвера `sds-local-volume` при управлении локальными томами на базе LVM.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
- **webhook** — сайдкар-контейнер, реализующий вебхук-сервер для проверки кастомных ресурсов LocalStorageClass и ресурсов StorageClass, а также для изменения значения поля `spec.schedulerName` у подов, использующих тома, созданные provisioner'ом `local.csi.storage.deckhouse.io`. После изменения значением поля `spec.schedulerName` становится `sds-local-volume`, что позволяет передать планирование таких подов компоненту `sds-local-volume-scheduler-extender` вместо

стандартного планировщика Kubernetes (kube-scheduler).

2. **Sds-local-volume-scheduler-extender** — состоит из одного контейнера, представляет собой расширение (extender) для kube-scheduler, реализует специфичную для подов, использующих локальные тома логику размещения. При планировании учитывается свободное место на узлах, используемых для размещения на них локальных томов, а также размер дискового пространства, которое надо зарезервировать под эти тома.
3. **CSI-драйвер (sds-local-volume)** — реализация CSI-драйвера для `local.csi.storage.deckhouse.io` provisioner. С типовой архитектурой CSI-драйвера, используемого в DKP, можно ознакомиться в разделе документации архитектуры CSI-драйвера. CSI-драйвер (sds-local-volume) — разработка компании Флант.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolume, PersistentVolumeClaim, VolumeAttachment, StorageClass;
- работа с кастомными ресурсами LocalStorageClass;
- создание ресурса StorageClass.

С модулем взаимодействуют следующие внешние компоненты:

1. Kube-apiserver:

- валидация кастомных ресурсов LocalStorageClass, ресурсов StorageClass;
- изменение поля `spec.schedulerName` подов, использующих тома, созданные при помощи `local.csi.storage.deckhouse.io` provisioner.

2. **Kube-scheduler** — отправка на вебхук `sds-local-volume-scheduler-extender` запросов на планирование подов, в поле `spec.schedulerName` которых указано значение `sds-local-volume`.

Дополнительные ресурсы

- Документация модуля `sds-local-volume`

Модуль sds-node-configurator

Модуль `sds-node-configurator` управляет LVM на узлах кластера с помощью кастомных ресурсов Kubernetes и выполняет следующие операции:

- Автоматически обнаруживает блочные устройства и создает, обновляет или удаляет соответствующие ресурсы BlockDevice.
- Автоматически обнаруживает на узлах группы томов LVM с тегом `storage.deckhouse.io/enabled=true`, а также thin pool в их составе, и управляет соответствующими ресурсами LVMVolumeGroup. Если для обнаруженной группы томов ресурс LVMVolumeGroup отсутствует, модуль создает его автоматически.
- Сканирует физические тома LVM, входящие в управляемые группы томов. При увеличении базовых блочных устройств автоматически расширяет соответствующие физические тома LVM с помощью `pvresize`.
- Создает, расширяет и удаляет группы томов LVM на узле в соответствии с настройками ресурсов LVMVolumeGroup.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

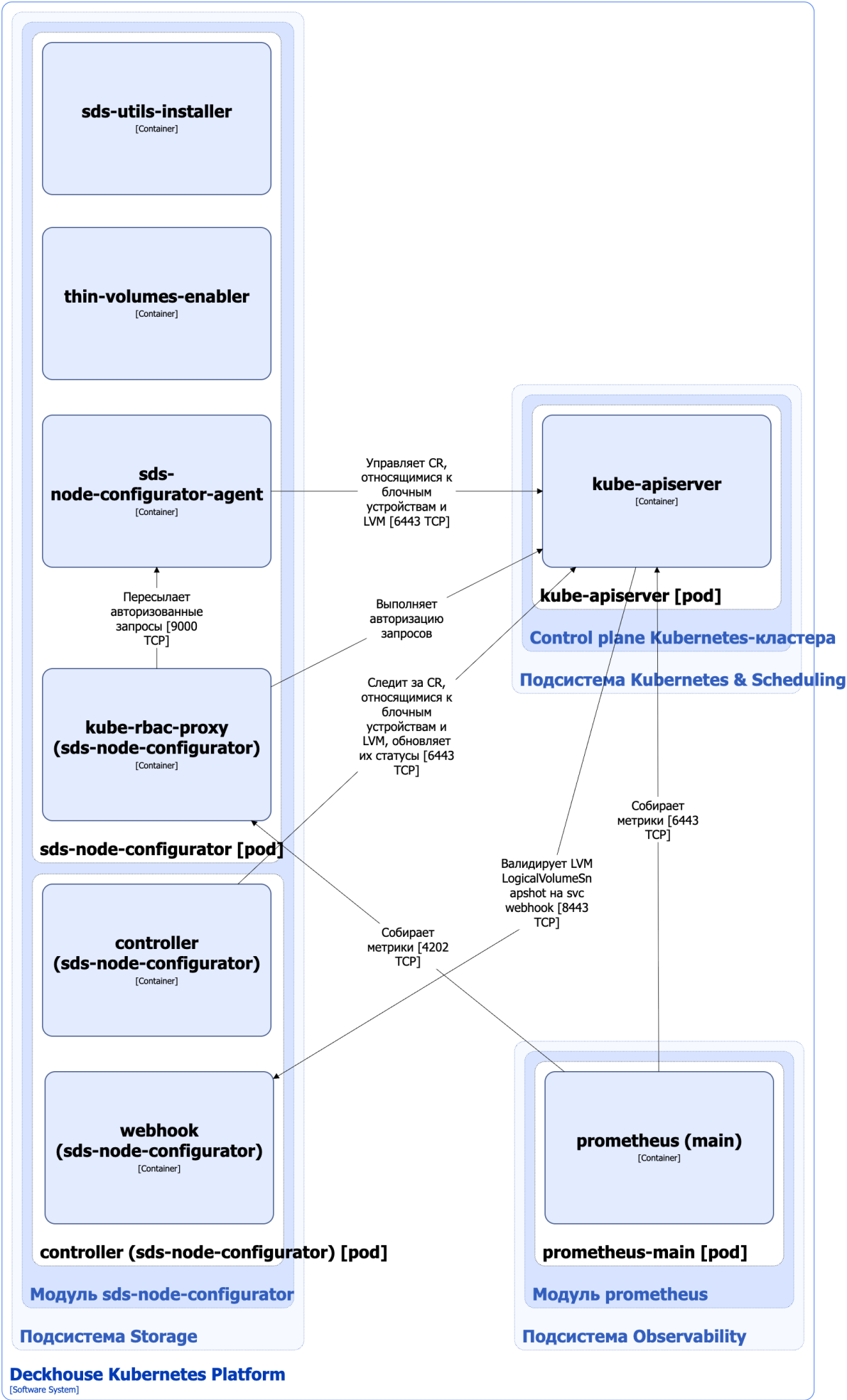
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `sds-node-configurator` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Sds-node-configurator** (DaemonSet) — контроллер, запущенный на узлах кластера и выполняющий перечисленные выше операции с кастомными ресурсами BlockDevice, LVMVolumeGroup, LVMLogicalVolume, LVMLogicalVolumeSnapshot и т.д. Полный список ресурсов, которыми управляет модуль, приведён в документации модуля.

Компонент содержит следующие контейнеры:

- **sds-utils-installer** — init-контейнер, устанавливающий набор утилит, необходимых для управления блочными устройствами и LVM-томами;
 - **thin-volumes-enabler** — init-контейнер, включающий поддержку thin томов;
 - **sds-node-configurator-agent** — основной контейнер;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контроллера. Является Open Source-проектом.
2. **Controller** (Deployment) — контроллер, следящий за кастомными ресурсами, относящимися к блочным устройствам и LVM. Controller работает с метаданными ресурсов и обновляет их статусы.

Компонент содержит следующие контейнеры:

- **controller** — основной контейнер;
- **webhook** — сайдкар-контейнер, реализующий вебхук-сервер для проверки кастомных ресурсов LVMLogicalVolumeSnapshot. Если используемая редакция DKP не поддерживает функционал снимков логических томов LVM, кастомный ресурс LVMLogicalVolumeSnapshot не проходит валидацию.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - для работы с кастомными ресурсами, относящимися к блочным устройствам и LVM;
 - для авторизации запросов к метрикам.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — выполняет валидацию кастомных ресурсов LVMLogicalVolumeSnapshot.
2. **Prometheus-main** — собирает метрики компонента `sds-node-configurator`.

Дополнительные ресурсы

- Документация модуля `sds-node-configurator`

Модуль csi-ceph

Модуль `csi-ceph` предназначен для интеграции DKP с Ceph-кластерами и обеспечивает управление хранилищем на основе RBD (RADOS Block Device) или CephFS. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса CephStorageClass.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

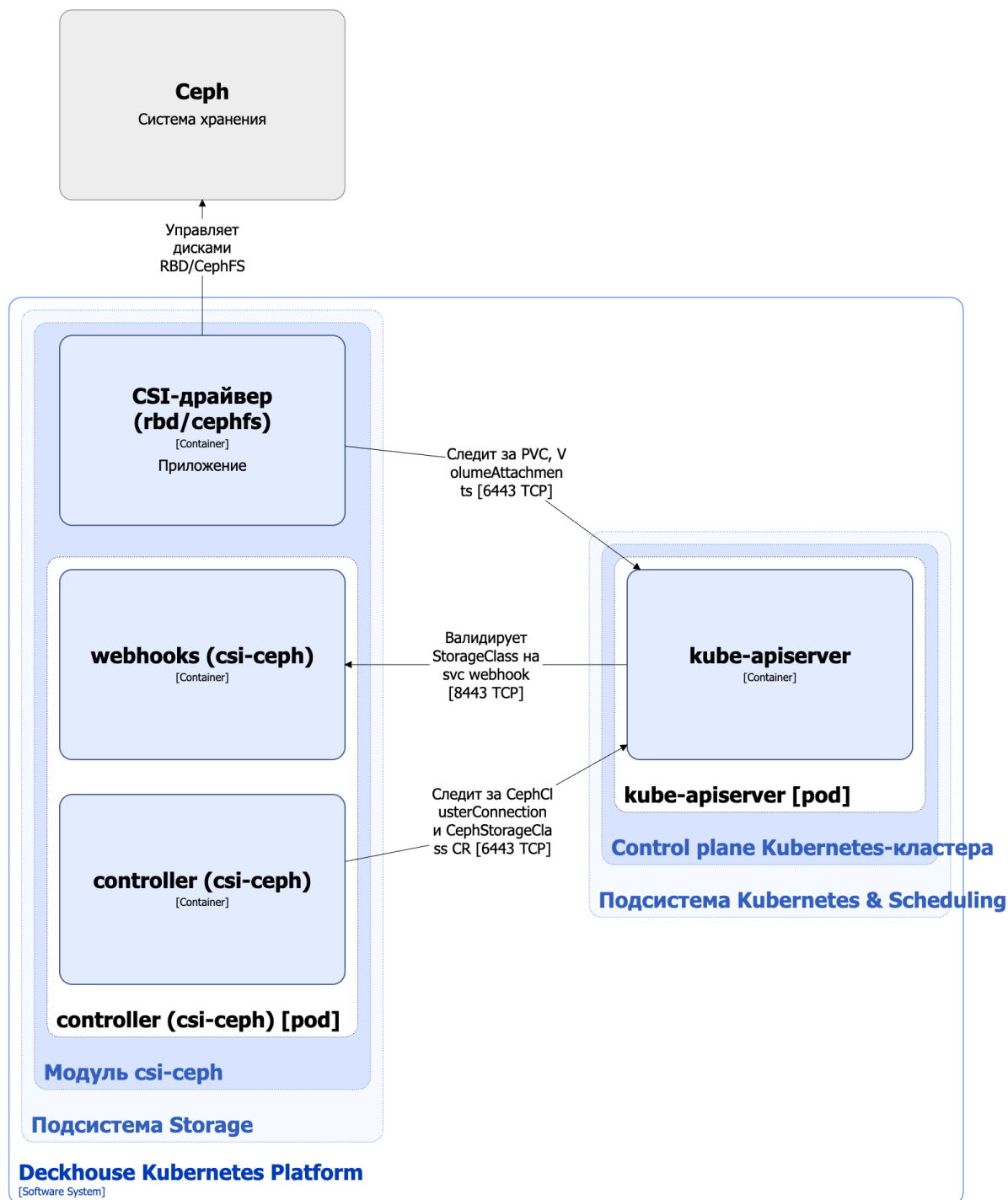
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-ceph` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:
 - CephClusterConnection — параметры подключения к кластеру Ceph;
 - CephStorageClass — определяет конфигурацию для создаваемого Kubernetes StorageClass.

В CephStorageClass задается тип storage-класса (CephFS, RBD), reclaim policy, параметры подключения к кластеру Ceph, а также специфичные для каждого storage-класса дополнительные параметры. В зависимости от типа storage-класса эти параметры используются provisioner'ом CSI-драйвера `rbd.csi.ceph.com` или `cephfs.csi.ceph.com` при управлении томами.

Кастомный ресурс CephMetadataBackup используется в сценариях миграции и восстановления, реализованных хуками модуля, а не основным runtime-контроллером.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
- **webhooks** — сайдкар-контейнер, реализующий вебхук-сервер для проверки стандартных ресурсов StorageClass.

2. **CSI-драйвер (rbd/cephfs)** — реализация CSI-драйвера для `rbd.csi.ceph.com` или `cephfs.csi.ceph.com` provisioner. Выбор CSI-драйвера выполняется путём задания storage-класса в кастомном ресурсе CephStorageClass.

CSI-драйвер `csi-cephfs` реализован по типовой архитектуре CSI-драйвера, используемого в DKP.

CSI-драйвер `csi-rbd` реализован по отличной от типового CSI-драйвера архитектуре.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

- **Kube-apiserver:**
 - мониторинг ресурсов PersistentVolume, PersistentVolumeClaim, VolumeAttachment и StorageClass;
 - работа с кастомными ресурсами CephClusterConnection и CephStorageClass;
 - создание ресурса StorageClass.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация ресурсов StorageClass.

Дополнительные ресурсы

- Документация модуля `csi-ceph`

Модуль csi-hpe

Модуль `csi-hpe` предназначен для управления томами с использованием систем хранения данных HPE. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса HPEStorageClass.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

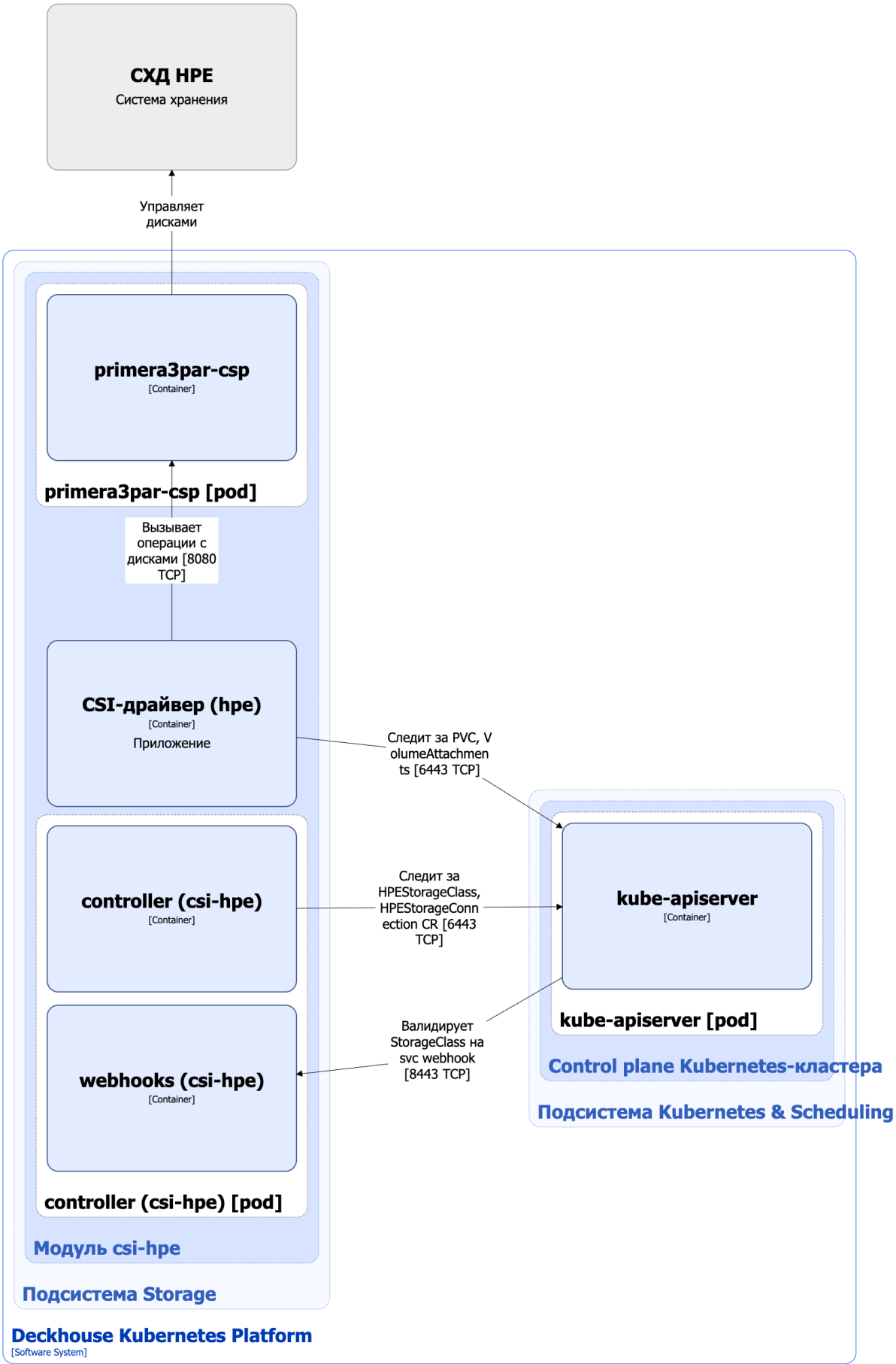
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-hpe` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

- 1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:

- `HPEStorageConnection` — параметры подключения к СХД HPE;
- `HPEStorageClass` — определяет конфигурацию для Kubernetes `StorageClass`.

В `HPEStorageClass` задается протокол подключения, название пула ресурсов, тип файловой системы и политика обработки тома при удалении PVC (`reclaim policy`).

Также controller синхронизирует лейбл `storage.deckhouse.io/csi-hpe-node` для узлов кластера в соответствии со значением селектора узлов `spec.settings.nodeSelector` кастомного ресурса `ModuleConfig`.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
 - **webhooks** — сайдкар-контейнер, реализующий вебхук-сервер для проверки `StorageClass`.
2. **CSI-драйвер (hpe)** — реализация CSI-драйвера, использующего provisioner `csi.hpe.com`. С типовой архитектурой CSI-драйвера, используемого в DKP, можно ознакомиться в описании архитектуры CSI-драйвера.
 3. **Primera3par-csp** — сервисный контейнер-провайдер (Container Storage Provider, CSP), обеспечивающий интеграцию CSI-драйвера с системами хранения данных HPE Primera и ZPAR. Выполняет функции взаимодействия между Kubernetes и СХД, а также управляет сессиями и репликацией путей.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов `PersistentVolume`, `PersistentVolumeClaim`, `VolumeAttachment` и `StorageClass`;
 - работа с кастомными ресурсами `HPEStorageConnection` и `HPEStorageClass`;
 - создание и обновление ресурсов `VolumeSnapshotClass`, `Secret` и `StorageClass`.
2. **Система хранения HPE** — создание, удаление и управление томами, а также организация мультипутевого доступа к данным.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация ресурсов `StorageClass`.

Дополнительные ресурсы

- Документация модуля `csi-hpe`

Модуль csi-huawei

Модуль `csi-huawei` предназначен для управления томами с использованием систем хранения данных Huawei. Он позволяет создавать `StorageClass` в Kubernetes с помощью ресурса `HuaweiStorageClass`.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

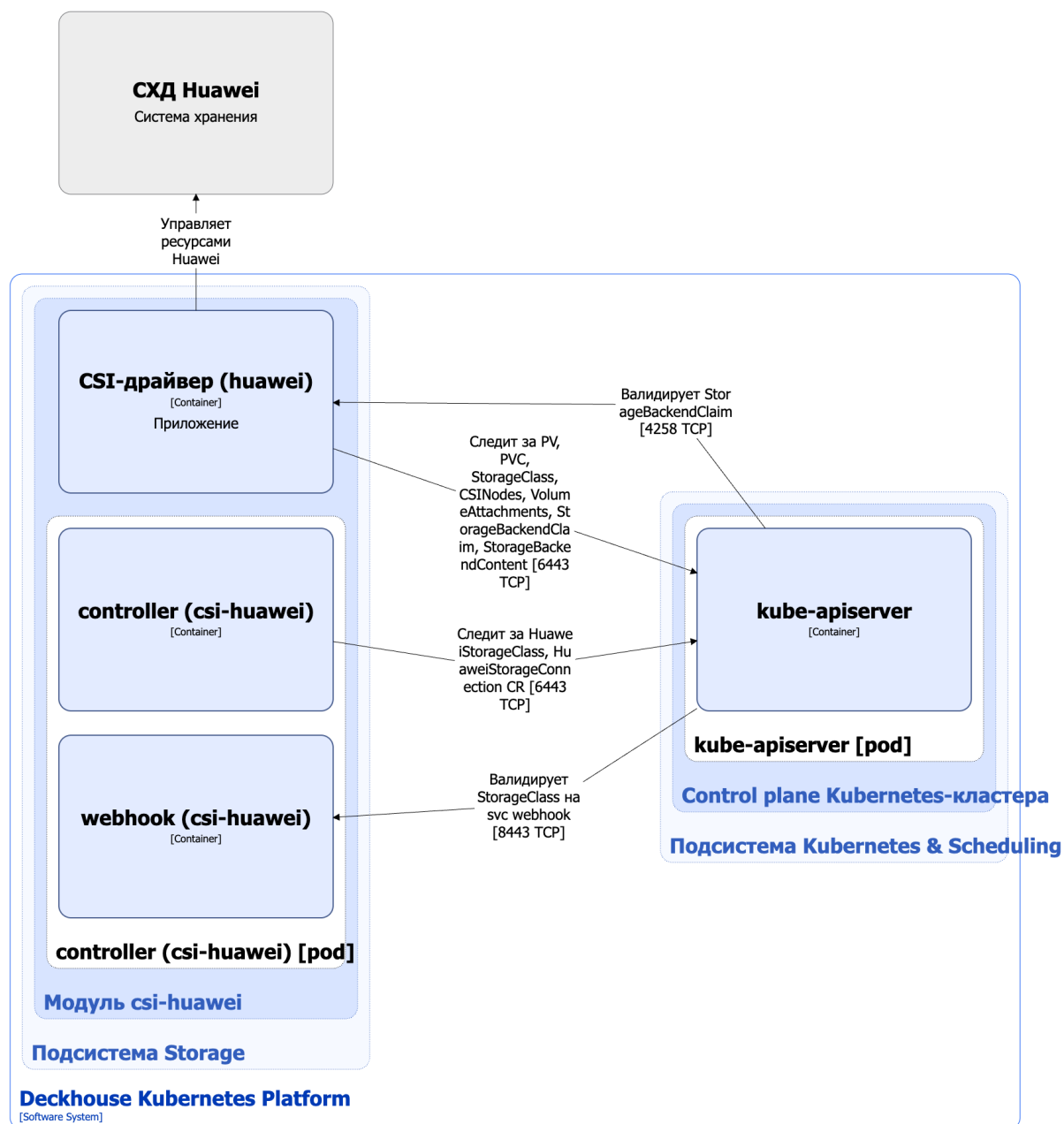
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-huawei` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:
 - HuaweiStorageConnection — параметры подключения к СХД Huawei;
 - HuaweiStorageClass — определяет конфигурацию для Kubernetes StorageClass.

В HuaweiStorageClass задается название пула ресурсов, тип файловой системы и политика обработки тома при удалении PVC (reclaim policy).

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
 - **webhooks** — сайдкар-контейнер, реализующий вебхук-сервер для валидации StorageClass.
2. **CSI-драйвер (huawei)** — реализация CSI-драйвера, использующего provisioner `csi.huawei.com`. С архитектурой CSI-драйвера `csi-huawei` можно ознакомиться в соответствующем разделе документации.

CSI-драйвер `csi-huawei` обслуживает следующие кастомные ресурсы:

- StorageBackendClaim — запрос на подключение к СХД Huawei;
- StorageBackendContent — описание фактического подключения к СХД Huawei.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolume, PersistentVolumeClaim, VolumeAttachment, StorageClass;
- работа с кастомными ресурсами HuaweiStorageConnection, HuaweiStorageClass, StorageBackendClaim, StorageBackendContent;
- создание ресурса StorageClass.

2. СХД Huawei — создание и удаление томов, подключение и отключение томов от узлов.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация кастомных ресурсов StorageBackendClaim, а также стандартных ресурсов StorageClass.

Дополнительные ресурсы

- Документация модуля `csi-huawei`

Модуль csi-netapp

Модуль `csi-netapp` предназначен для управления томами с использованием систем хранения данных NetApp. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса NetappStorageClass.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

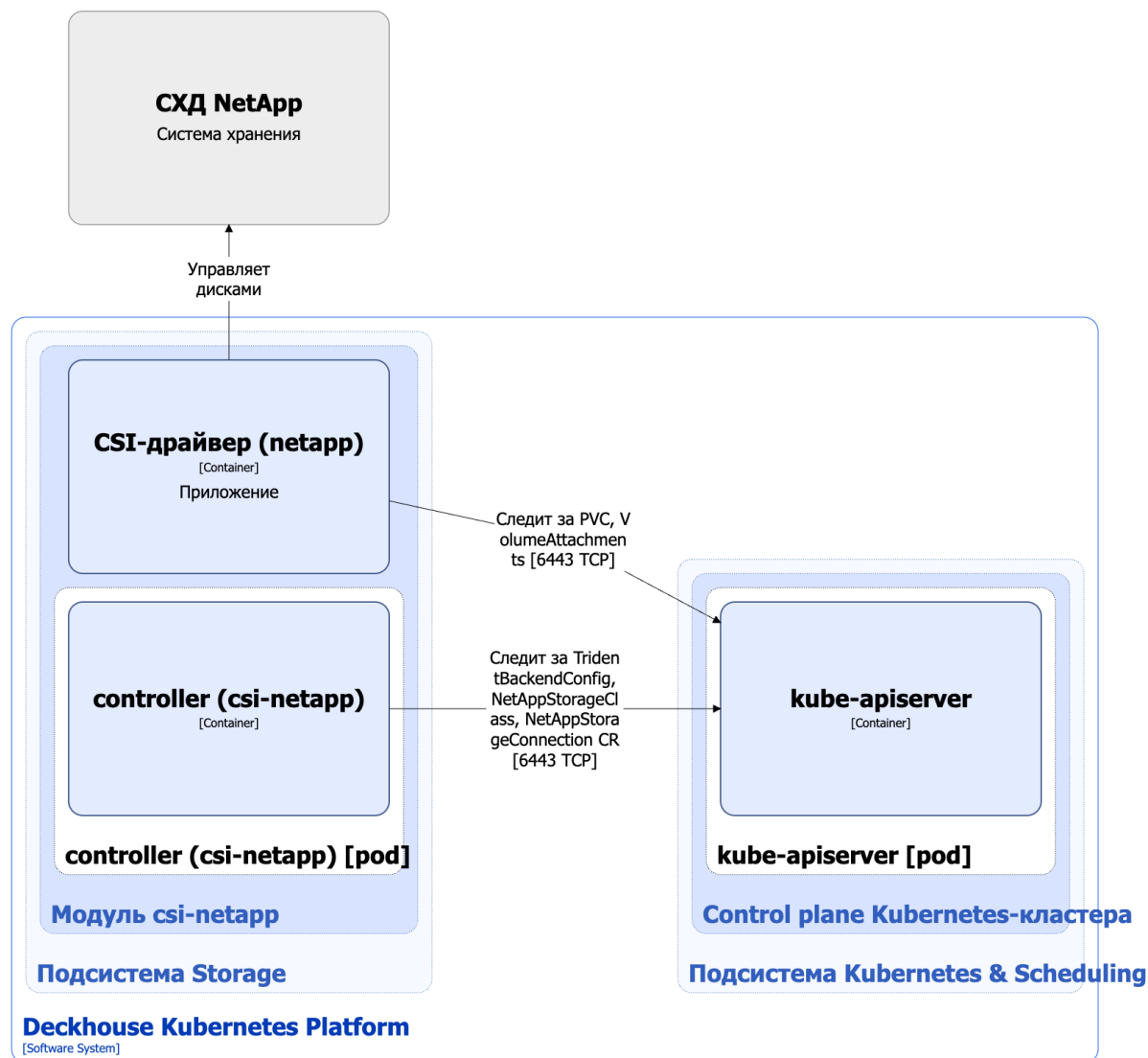
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-netapp` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:
 - `NetappStorageConnection` — параметры подключения к СХД NetApp;
 - `NetappStorageClass` — определяет конфигурацию для создания Kubernetes StorageClass, который использует provisioner `csi.trident.netapp.io`.

Также controller синхронизирует лейбл `storage.deckhouse.io/csi-netapp-node` для узлов кластера в соответствии со значением селектора узлов `spec.settings.nodeSelector` кастомного ресурса `ModuleConfig`.

Состоит из одного основного контейнера **controller**.

2. **CSI-драйвер (netapp)** — реализация CSI-драйвера, использующего provisioner `csi.trident.netapp.io`. С типовой архитектурой CSI-драйвера, используемого в DKP, можно ознакомиться в описании архитектуры CSI-драйвера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов `PersistentVolume`, `PersistentVolumeClaim`, `VolumeAttachment` и `StorageClass`;
 - работа с кастомными ресурсами `TridentBackendConfig`, `NetappStorageConnection` и `NetappStorageClass`;
 - создание ресурса `VolumeSnapshotClass`, `Secret` и `StorageClass`.

2. **СХД NetApp** — создание, удаление и управление томами, а также подключение и отключение томов от узлов.

Дополнительные ресурсы

- Документация модуля `csi-netapp`

Модуль csi-nfs

Модуль `csi-nfs` предназначен для управления NFS-томами. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса NFSStorageClass.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

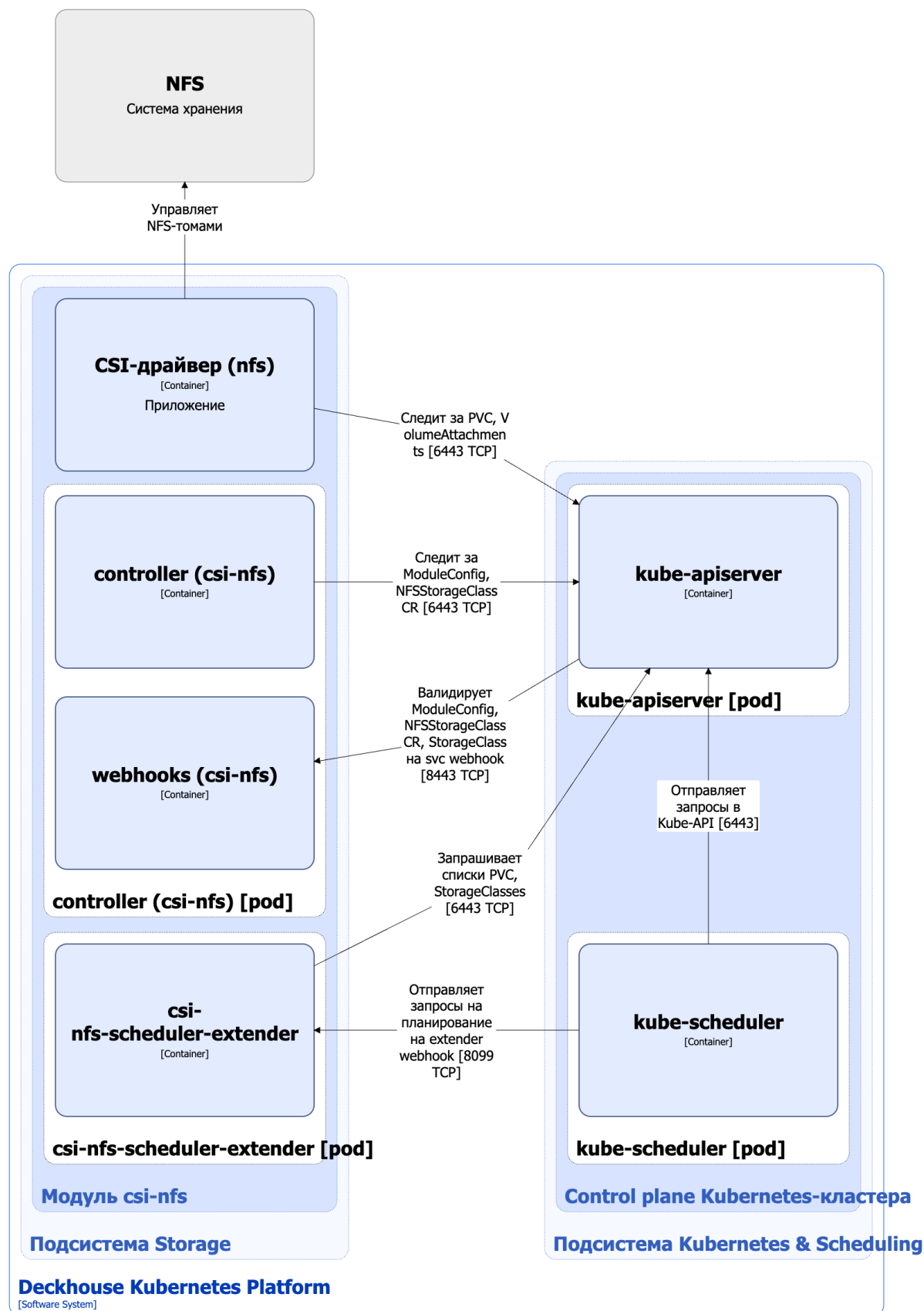
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-nfs` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий кастомные ресурсы NFSStorageClass. Ресурс NFSStorageClass определяет конфигурацию для создаваемого Kubernetes StorageClass, который использует provisioner `nfs.csi.k8s.io`.

В создаваемом StorageClass задаются параметры подключения к NFS-серверу, reclaim policy, volume binding mode и другие параметры. Эти параметры затем использует provisioner CSI-

драйвера `csi-nfs` при управлении NFS-томами.

Также controller устанавливает лейбл `storage.deckhouse.io/csi-nfs-node` для узлов кластера в соответствии со значением параметра `spec.workloadNodes.nodeSelector` кастомного ресурса `NFSStorageClass`.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
 - **webhooks** — сайдкар-контейнер, реализующий вебхук-сервер для проверки кастомных ресурсов `ModuleConfig`, `NFSStorageClass` и стандартных ресурсов `StorageClass`.
2. **Csi-nfs-scheduler-extender** — состоит из одного контейнера и представляет собой расширение (extender) для `kube-scheduler`. Реализует специфичную для подов логику размещения при использовании NFS-томов. При планировании подов учитываются селекторы узлов, заданные в кастомном ресурсе `NFSStorageClass`.
- Компонент может отсутствовать если селекторы узлов в кастомном ресурсе `NFSStorageClass` не заданы.
3. **CSI-драйвер (csi-nfs)** — реализация CSI-драйвера для provisioner `nfs.csi.k8s.io` (NFS CSI driver). С архитектурой CSI-драйвера `csi-nfs` можно ознакомиться на странице описания CSI-драйвера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
- мониторинг ресурсов `PersistentVolume`, `PersistentVolumeClaim`, `VolumeAttachment`, `StorageClass`;
 - работа с кастомными ресурсами `NFSStorageClass`;
 - создание ресурса `StorageClass`.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — валидация кастомных ресурсов `ModuleConfig` и `NFSStorageClass`, а также стандартных ресурсов `StorageClass`.
2. **Kube-scheduler** — отправка на вебхук `csi-nfs-scheduler-extender` запросов на планирование подов, использующих NFS-тома.

Дополнительные ресурсы

- Документация модуля `csi-nfs`

Модуль csi-s3

Модуль `csi-s3` предназначен для управления томами на основе S3-хранилищ. Модуль использует `geeseFS` — файловую систему FUSE на основе S3. Модуль позволяет создавать `StorageClass` в Kubernetes с помощью ресурса `S3StorageClass`.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

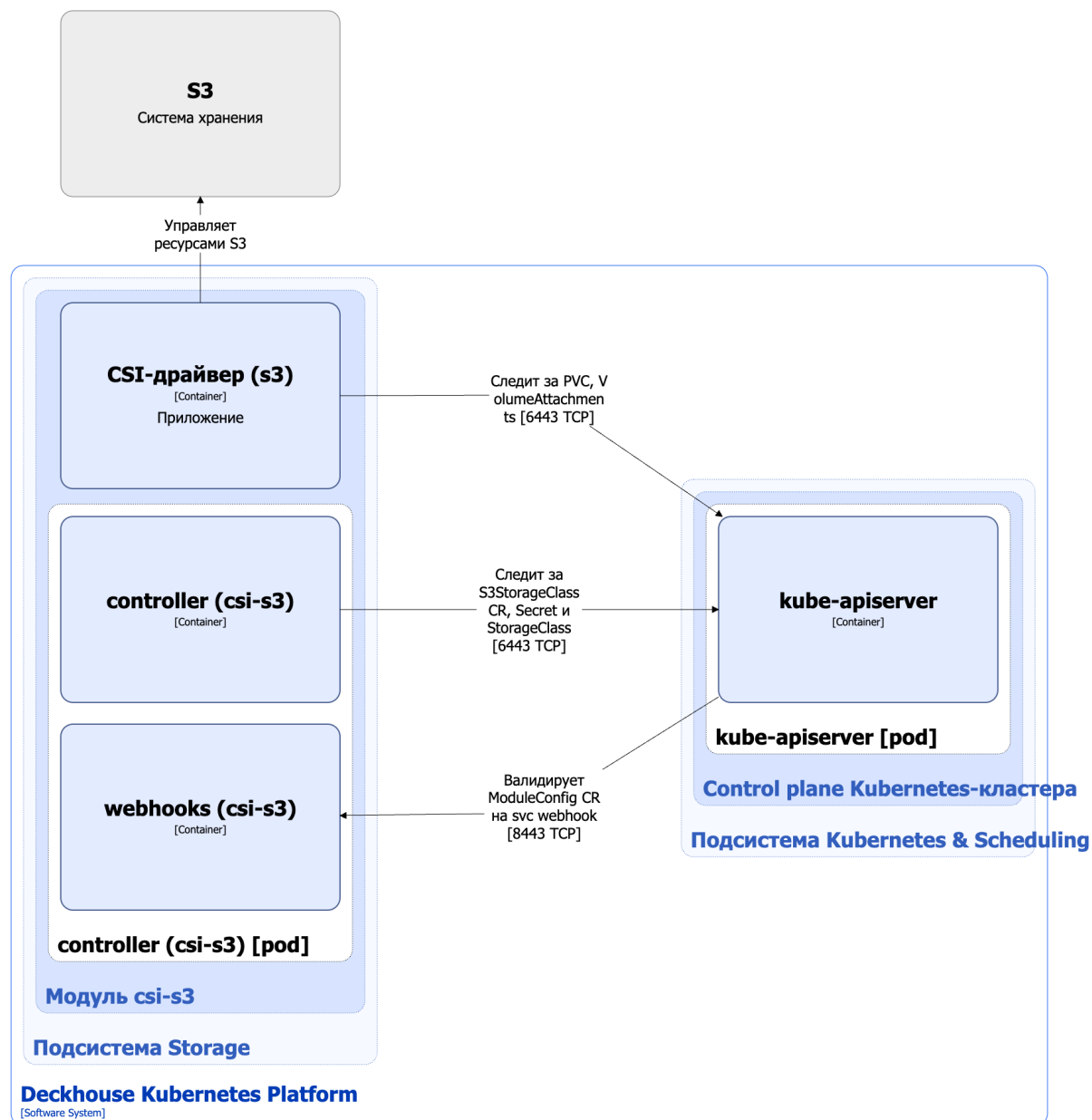
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-s3` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий кастомный ресурс `S3StorageClass`. Ресурс `S3StorageClass` определяет конфигурацию для создаваемого Kubernetes StorageClass, использующего provisioner `ru.yandex.s3.csi`.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
 - **webhooks** — сайдкар-контейнер, реализующий вебхук-сервер для проверки кастомного ресурса `ModuleConfig`.
2. **CSI-драйвер (s3)** — реализация CSI-драйвера, использующего provisioner `ru.yandex.s3.csi`. С архитектурой CSI-драйвера `csi-s3` можно ознакомиться на странице описания CSI-драйвера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов `PersistentVolume`, `PersistentVolumeClaim`, `VolumeAttachment`, `Secret` и `StorageClass`;
 - работа с кастомными ресурсами `S3StorageClass` и `ModuleConfig`;

- создание ресурсов StorageClass и Secret.

2. **S3-хранилище** — создание и удаление томов, подключение и отключение томов от узлов.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация кастомных ресурсов ModuleConfig.

Дополнительные ресурсы

- Документация модуля `csi-s3`

Модуль csi-scsi-generic

Модуль `csi-scsi-generic` предназначен для управления томами на СХД с подключением по SCSI. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса SCSIStorageClass и поддерживает работу как по технологии iSCSI, так и по Fibre Channel (FC).

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

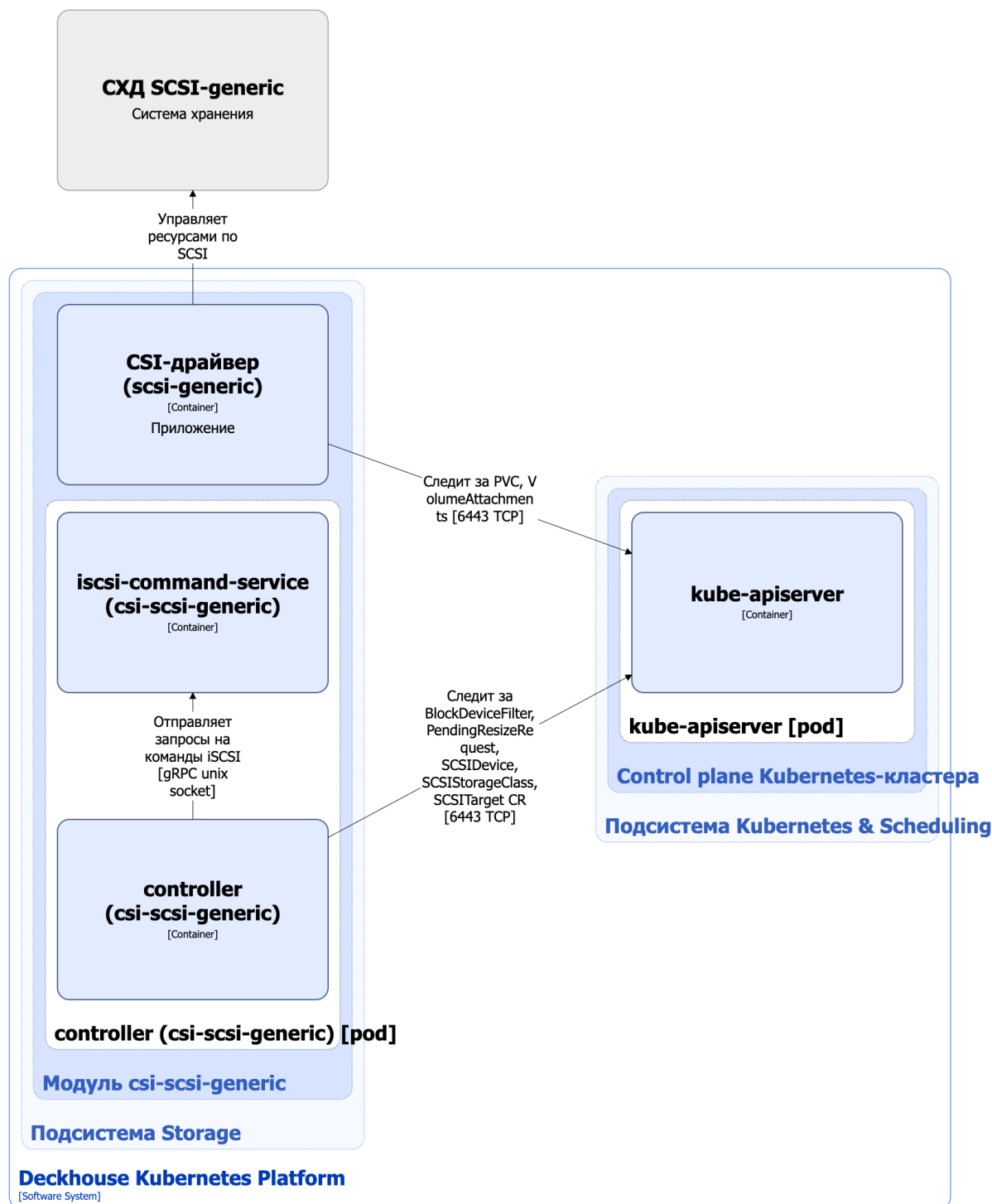
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-scsi-generic` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:
 - **SCSTarget** — описание точки подключения к CXД (iSCSI/FC);
 - **SCSIDevice** — описание обнаруженного SCSI-устройства;
 - **PendingResizeRequest** — заявка на отложенное расширение PVC, если запрошенный размер больше текущего размера устройства;
 - **SCSIStorageClass** — определяет конфигурацию для Kubernetes StorageClass;
 - **BlockDeviceFilter** — определяет фильтры для выбора физических SCSI-устройств, которые можно использовать для создания томов системой. Позволяет гибко управлять пулом доступных устройств путем задания правил по атрибутам устройств и исключения/разрешения отдельных устройств.

В **SCSIStorageClass** задаются селектор устройств (**scsiDeviceSelector**), политика обработки тома при удалении PVC (**reclaim policy**) и параметры очистки тома.

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
 - **iscsi-command-service** — сайдкар-контейнер, реализующий обнаружение SCSI-устройств.
2. **CSI-драйвер (csi-scsi-generic)** — реализация CSI-драйвера, использующего provisioner `scsi-generic.csi.storage.deckhouse.io`. С архитектурой CSI-драйвера `csi-scsi-generic` можно ознакомиться в соответствующем разделе документации.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов PersistentVolume, PersistentVolumeClaim, VolumeAttachment, StorageClass;
 - работа с кастомными ресурсами BlockDeviceFilter, SCSTarget, SCSIDevice, PendingResizeRequest и SCSIStorageClass;
 - создание ресурса StorageClass.
2. **СХД с подключением по SCSI** — координация использования уже доступных SCSI-устройств, их привязка, очистка и подключение на узлах.

Дополнительные ресурсы

- Документация модуля `csi-scsi-generic`

Модуль csi-vsphere

Модуль `csi-vsphere` предоставляет поддержку Container Storage Interface (CSI) для сред VMware vSphere, обеспечивая динамическое предоставление и управление постоянными томами хранения в кластерах Kubernetes, работающих на инфраструктуре vSphere.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

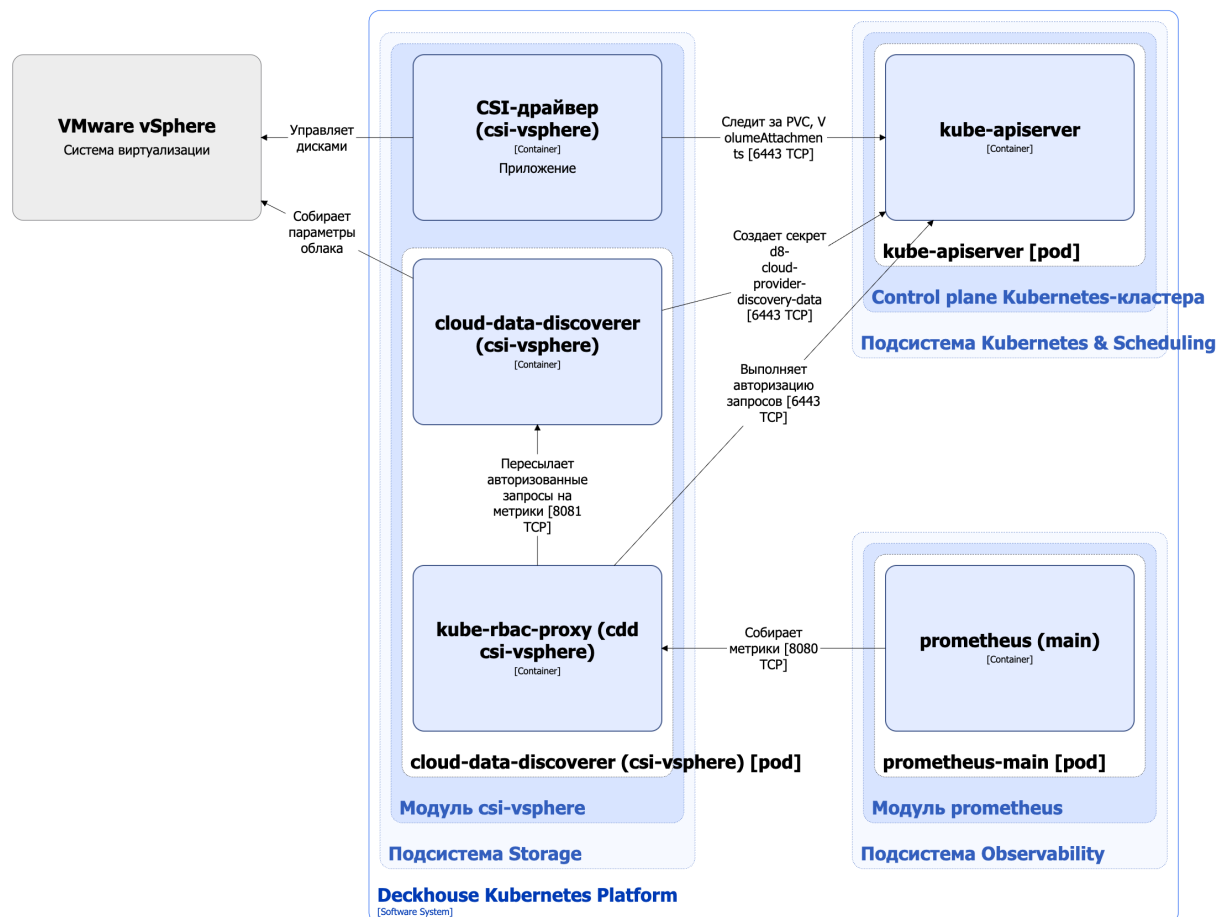
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-vsphere` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Cloud-data-discoverer** — отвечает за сбор данных из API VMware vSphere и предоставление их в виде секрета `kube-system/d8-cloud-provider-discovery-data`. Этот секрет содержит параметры конкретного облака, которые используются CSI-драйвером для управления томами.

Состоит из следующих контейнеров:

- **cloud-data-discoverer** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера `cloud-data-discoverer`.

2. **CSI-драйвер (vsphere)** — реализация CSI-драйвера для VMware vSphere. С архитектурой CSI-драйвера, используемого в модуле `csi-vsphere` DKP, можно ознакомиться в соответствующем разделе документации.

CSI-драйвер (vsphere) не поддерживает работу со снимками. По этой причине в поде `csi-controller` отсутствует сайдкар-контейнер `snapshotter` (`external-snapshotter`).

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - мониторинг ресурсов `PersistentVolumeClaim`, `VolumeAttachment`;
 - создание секрета `kube-system/d8-cloud-provider-discovery-data`;
 - создание `StorageClass` на основе результатов обнаружения;
 - авторизация запросов на получение метрик.
2. **VMware vSphere:**
 - получение параметров облака;
 - управление дисками.

С модулем взаимодействуют следующие внешние компоненты:

- **Prometheus-main** — сбор метрик `cloud-data-discoverer`.

Дополнительные ресурсы

- Документация модуля `csi-vsphere`

Модуль `csi-yadro-tatlin-unified`

Модуль `csi-yadro-tatlin-unified` предназначен для управления томами с использованием систем хранения данных TATLIN.UNIFIED. Он позволяет создавать StorageClass в Kubernetes с помощью ресурса `YadroTatlinUnifiedStorageClass`.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

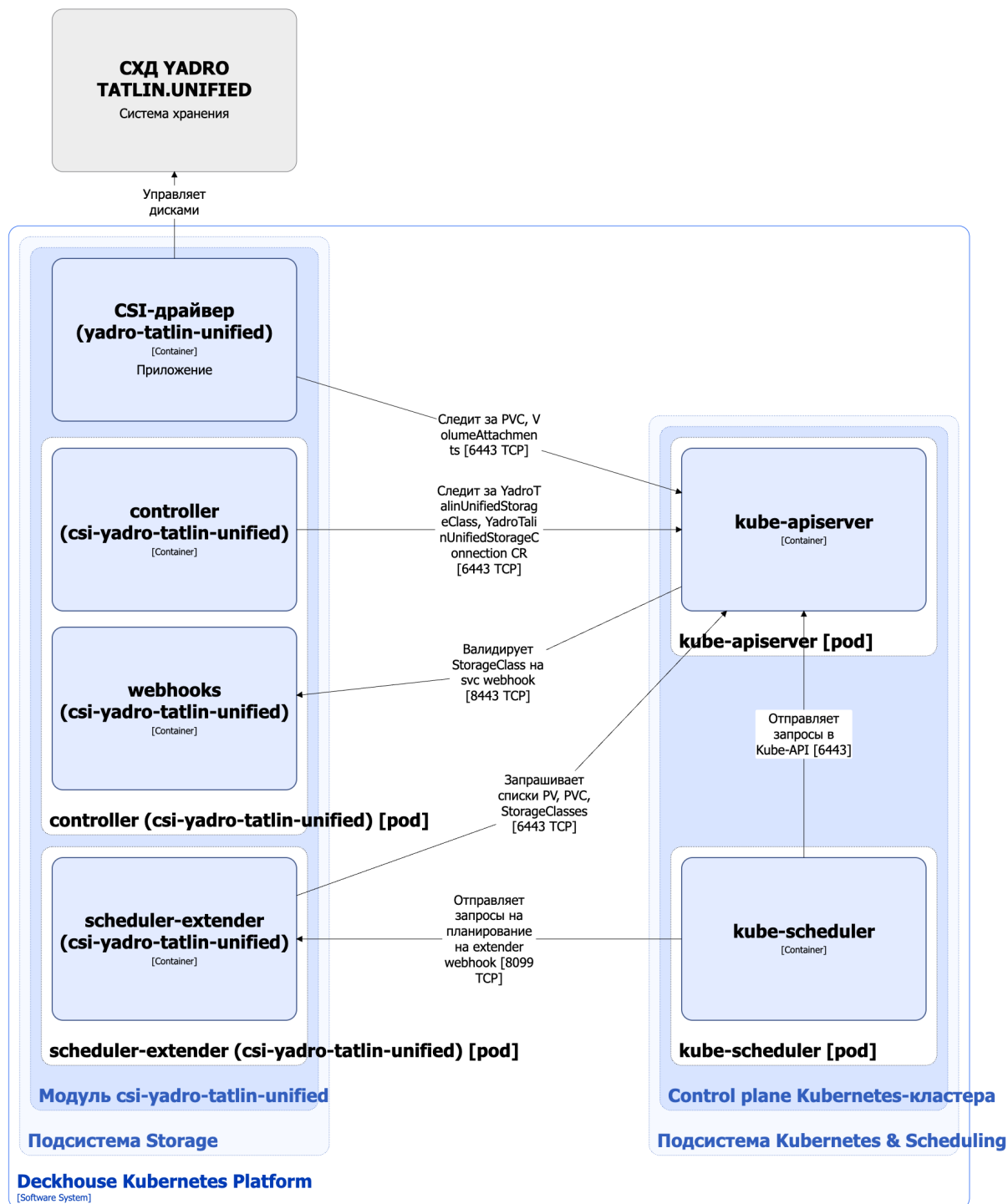
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `csi-yadro-tatlin-unified` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Controller** — контроллер, обслуживающий следующие кастомные ресурсы:
 - `YadroTatlinUnifiedStorageConnection` — параметры подключения к СХД TATLIN.UNIFIED;
 - `YadroTatlinUnifiedStorageClass` — определяет конфигурацию для создаваемого Kubernetes StorageClass, который использует provisioner `csi-tatlinunified.yadro.com`.

В `YadroTatlinUnifiedStorageClass` задаются параметры подключения, а также название пула ресурсов, тип файловой системы и политика обработки тома при удалении PVC (reclaim policy).

Состоит из следующих контейнеров:

- **controller** — основной контейнер;
- **webhook** — сайдкар-контейнер, реализующий вебхук-сервер для проверки ресурсов StorageClass.

2. **CSI-драйвер (yadro-tatlin-unified)** — реализация CSI-драйвера, использующего provisioner `csi-tatlinunified.yadro.com`. С типовой архитектурой CSI-драйвера, используемого в DKP, можно ознакомиться на странице описания CSI-драйвера.
3. **Scheduler-extender** — состоит из одного контейнера, представляет собой расширение (extender) для kube-scheduler. Реализует специфичную для подов логику размещения при использовании томов СХД TATLIN.UNIFIED. При планировании подов учитываются селекторы узлов, заданные в кастомном ресурсе `YadroTatlinUnifiedStorageConnection` в параметрах `controlPlane` и `dataPlane`.

Компонент может отсутствовать, если селекторы узлов в кастомном ресурсе `YadroTatlinUnifiedStorageConnection` не заданы.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - мониторинг ресурсов `PersistentVolume`, `PersistentVolumeClaim`, `VolumeAttachment`, `StorageClass`;
 - работа с кастомными ресурсами `YadroTatlinUnifiedStorageConnection`, `YadroTatlinUnifiedStorageClass`;
 - создание ресурса `StorageClass`.
2. **СХД YADRO TATLIN.UNIFIED** — создание и удаление томов, подключение и отключение томов от узлов.

С модулем взаимодействуют следующие внешние компоненты:

- **Kube-apiserver** — валидация стандартных ресурсов `StorageClass`.

Дополнительные ресурсы

- Документация модуля `csi-yadro-tatlin-unified`

CSI-драйвер (типовой)

Для управления постоянными томами хранения в Deckhouse Kubernetes Platform (DKP) используется CSI-драйвер (плагин).

Container Storage Interface (CSI) — это стандартный интерфейс, который унифицирует доступ к хранилищам и упрощает интеграцию различных систем хранения в кластеры.

CSI-драйвер входит в состав модулей `cloud-provider-*` и `csi-*` DKP. Хотя для каждого поддерживаемого облачного провайдера или системы хранения используется собственная реализация спецификации CSI, архитектура CSI-драйвера у всех реализаций одинакова. Реализации могут для некоторых модулей отличаться по составу возможностей (capabilities) и компонентов.

Далее описана типовая архитектура CSI-драйвера, используемая в DKP. Она включает все возможные компоненты и реализует всю функциональность, применяемую в модулях DKP.

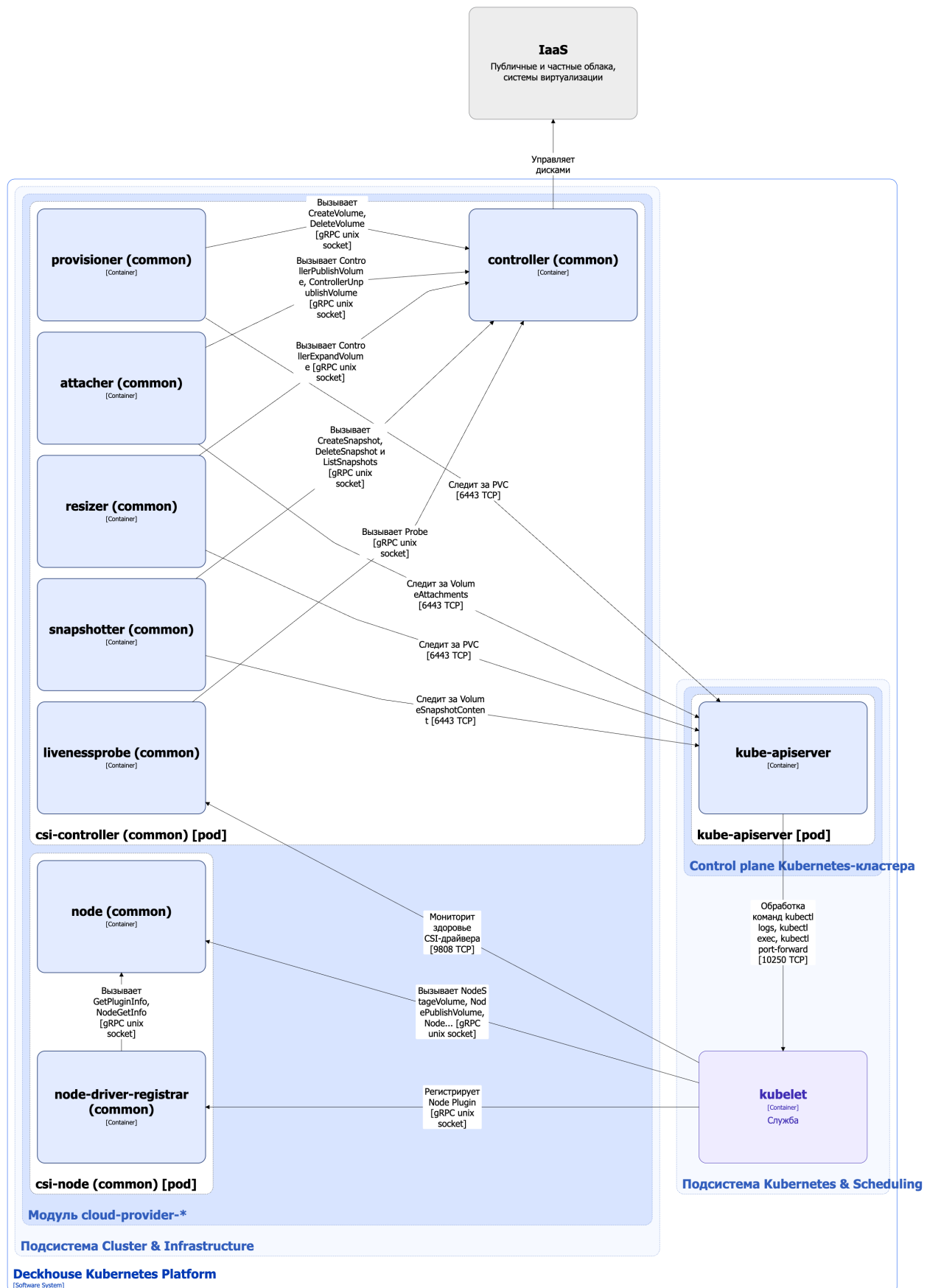
Архитектура драйвера

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Типовая архитектура CSI-драйвера на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов, а также управление снимками. Например, в AWS этот компонент вызывает EC2 API для создания томов EBS.

Состоит из следующих контейнеров:

- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно спецификации CSI;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC CreateVolume или DeleteVolume. Также использует RPC ValidateVolumeCapabilities для проверки совместимости;
 - **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC ControllerPublishVolume и ControllerUnpublishVolume;
 - **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC ControllerExpandVolume, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability EXPAND_VOLUME;
 - **snapshotter** (external-snapshotter) — работает совместно с модулем snapshot-controller, следит за ресурсами VolumeSnapshotContent, а также управляет снимками томов через RPC CreateSnapshot, DeleteSnapshot и ListSnapshots (если драйвер это поддерживает);
 - **livenessprobe** — отслеживает состояние CSI-драйвера через RPC Probe из Identity Service и предоставляет HTTP-эндпоинт /healthz, за которым следит kubelet. При неуспешной livenessProbe kubelet перезапускает под csi-controller.
2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability CAP_SYS_ADMIN. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC GetPluginInfo и NodeGetInfo, чтобы получить информацию о плагине и узле. Взаимодействуют с контейнером **node** по gRPC через Unix-сокет.



Некоторые сайдкар-контейнеры из списка внешних контроллеров, (например, snapshotter) могут отсутствовать в Deployment определенных cloud-provider-* или csi-*, если соответствующая функциональность не поддерживается реализацией CSI-драйвера.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment и VolumeSnapshotContent.
2. **Облачная инфраструктура** (или система виртуализации) — создание и удаление томов, подключение и отключение томов от узлов, управление снимками.

С драйвером взаимодействуют следующие внешние компоненты:

1. Kubelet:

- проверяет livenessProbe CSI-драйвера;
- регистрирует Node Plugin;
- вызывает RPC `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume` и `NodeExpandVolume` в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокеты.

Дополнительные ресурсы

- Документация модуля `snapshot-controller`

CSI-драйвер Ceph RBD

CSI-драйвер `csi-rbd` — это реализация стандарта Container Storage Interface (CSI) для обеспечения работы с Ceph RBD (RADOS Block Device) томами в Deckhouse Kubernetes Platform (DKP).

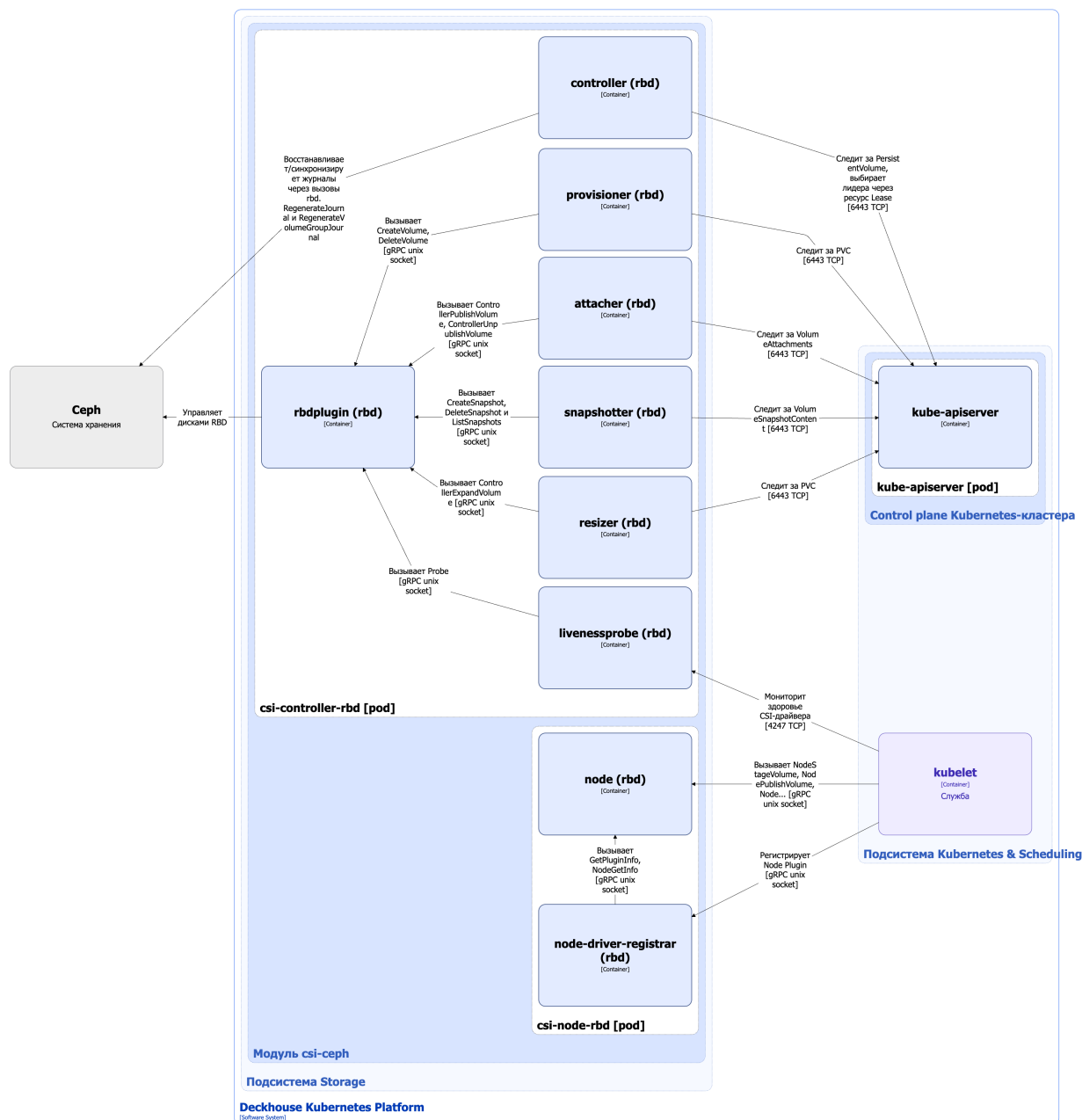
Архитектура CSI-драйвера

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера `csi-rbd` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер `csi-rbd` состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов, а также управление снимками.

Состоит из следующих контейнеров:

- **controller** — основной контейнер, который выполняет фоновые задачи управления ресурсами Kubernetes, связанными с Ceph.
- **rbdpugin** — сайдкар-контейнер, предоставляющий CSI endpoint через Unix-сокеты и реализующий gRPC-сервисы Identity Service и Controller Service согласно спецификации CSI. Он выполняет операции с RBD-томами в кластере Ceph;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие CSI RPC-методы, предоставляемые контейнером rbdplugin. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером rbdplugin по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC `CreateVolume` или `DeleteVolume`. Также использует RPC `ValidateVolumeCapabilities` для проверки совместимости;
 - **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC `ControllerPublishVolume` и `ControllerUnpublishVolume`;
 - **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC `ControllerExpandVolume`, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability `EXPAND_VOLUME`;
 - **snapshotter** (external-snapshotter) — работает совместно с модулем `snapshot-controller`, следит за ресурсами VolumeSnapshotContent, а также управляет снимками томов через RPC `CreateSnapshot`, `DeleteSnapshot` и `ListSnapshots` (если драйвер это поддерживает);
 - **livenessprobe** — отслеживает состояние CSI-драйвера через RPC `Probe` из Identity Service и предоставляет HTTP-эндпоинт `/healthz`, за которым следит kubelet. При неуспешной `livenessProbe` kubelet перезапускает под csi-controller.
2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability `CAP_SYS_ADMIN`. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC `GetPluginInfo` и `NodeGetInfo`, чтобы получить информацию о плагине и узле. Взаимодействуют с контейнером **node** по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment и VolumeSnapshotContent.
2. **Кластер Ceph** — создание и удаление томов, подключение и отключение томов от узлов, управление снимками.

С драйвером взаимодействуют следующие внешние компоненты:

- Kubelet:
 - проверяет `livenessProbe` CSI-драйвера;
 - регистрирует Node Plugin;
 - вызывает RPC `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume` и `NodeExpandVolume` в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокет.

Дополнительные ресурсы

- Документация модуля `snapshot-controller`

CSI-драйвер Huawei

CSI-драйвер `csi-huawei` — это реализация стандарта Container Storage Interface (CSI) для работы с томами на СХД Huawei в Deckhouse Kubernetes Platform (DKP).

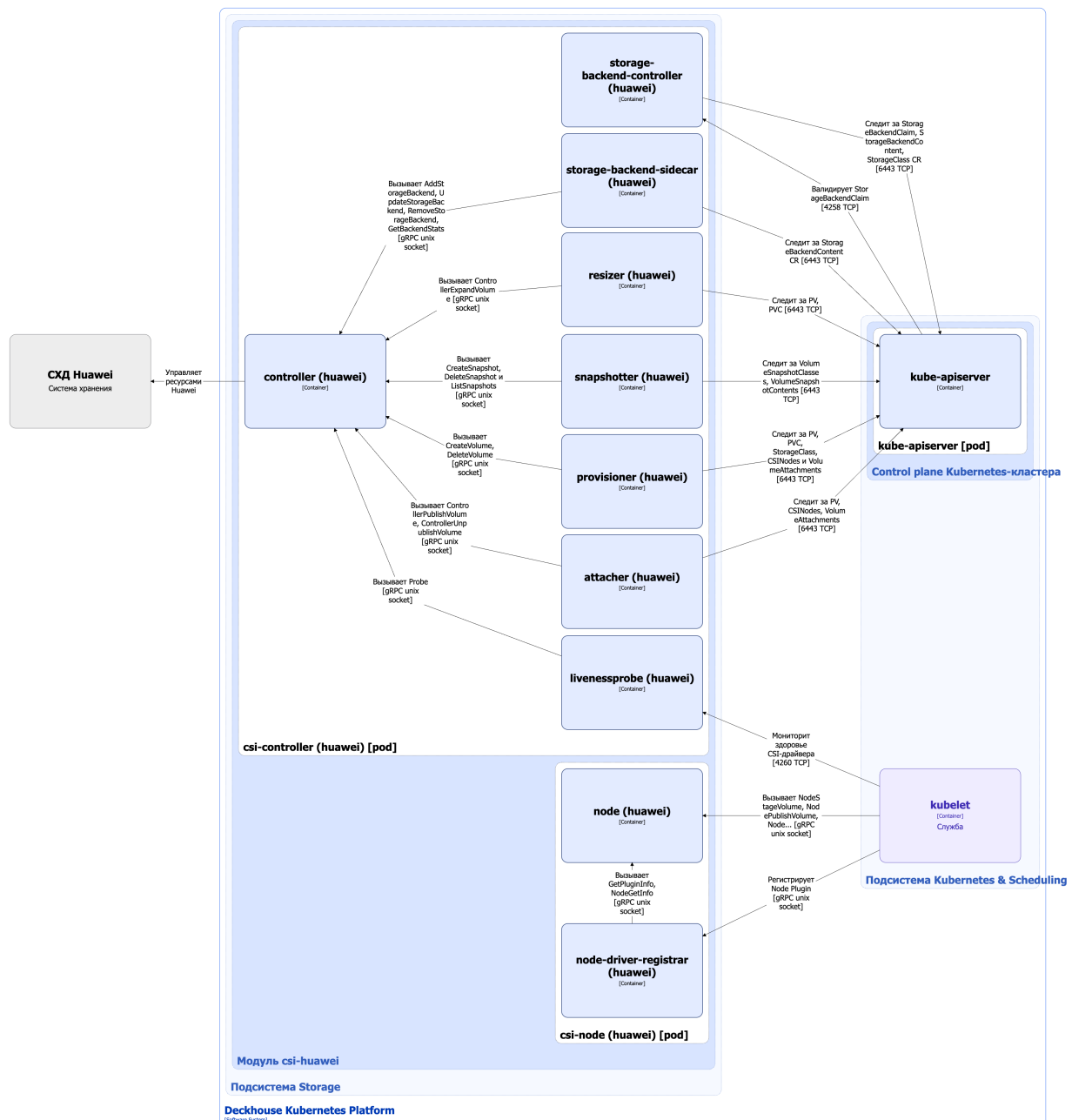
Архитектура драйвера

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера `csi-huawei` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер `csi-huawei` состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов.

Состоит из следующих контейнеров:

- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно спецификации CSI;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC CreateVolume или DeleteVolume. Также использует RPC ValidateVolumeCapabilities для проверки совместимости;
- **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC ControllerPublishVolume и ControllerUnpublishVolume;
- **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC ControllerExpandVolume, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability EXPAND_VOLUME;
- **snapshotter** (external-snapshotter) — работает совместно с модулем snapshot-controller, следит за ресурсами VolumeSnapshotContent, а также управляет снимками томов через RPC CreateSnapshot, DeleteSnapshot и ListSnapshots;
- **livenessprobe** — отслеживает состояние CSI-драйвера через RPC Probe из Identity Service и предоставляет HTTP-эндпоинт /healthz, за которым следит kubelet. При неуспешной livenessProbe kubelet перезапускает под csi-controller.
- **storage-backend-controller** — дополнительный контроллер, обеспечивающий работу со следующими кастомными ресурсами:
 - StorageBackendClaim — запрос на подключение к СХД Huawei;
 - StorageBackendContent — описание фактического подключения к СХД Huawei.
- **storage-backend-sidecar** — сайдкар-контейнер, обеспечивающий работу с СХД Huawei через вызовы RPC AddStorageBackend, UpdateStorageBackend, RemoveStorageBackend, GetBackendStats.

2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability CAP_SYS_ADMIN. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC GetPluginInfo и NodeGetInfo, чтобы получить информацию о плагине и узле. Взаимодействует с контейнером **node** по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment;

- работа с кастомными ресурсами `StorageBackendClaim`, `StorageBackendContent`.

2. **СХД Huawei** — создание и удаление томов, подключение и отключение томов от узлов.

С драйвером взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — валидация кастомных ресурсов `StorageBackendClaim`;

2. Kubelet:

- проверяет `livenessProbe` CSI-драйвера;
- регистрирует `Node Plugin`;
- вызывает RPC `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume` и `NodeUnpublishVolume` в `Node Plugin`.

Kubelet взаимодействует с `Node Plugin` по gRPC через Unix-сокеты.

Дополнительные ресурсы

- Документация модуля `snapshot-controller`

CSI-драйвер NFS

CSI-драйвер `csi-nfs` — это реализация стандарта Container Storage Interface (CSI) для управления NFS-томами в Deckhouse Kubernetes Platform (DKP).

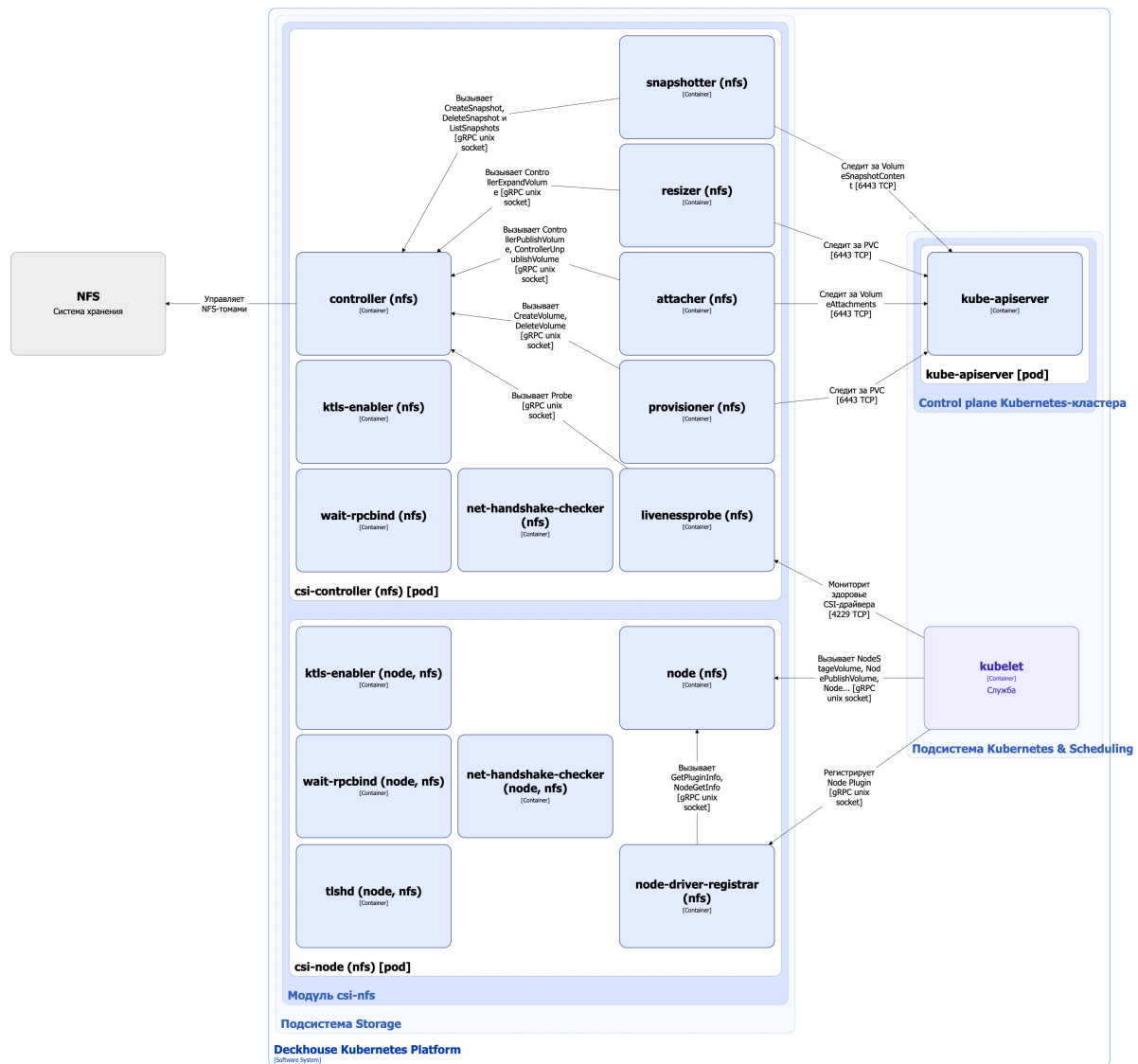
Архитектура драйвера

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера `csi-nfs` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер `csi-nfs` состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов, а также управление снимками.

Состоит из следующих контейнеров:

- **wait-rpcbind** — init-контейнер, ожидающий доступности Unix-сокета службы `rpcbind`;
- **ktlis-enabler** — init-контейнер, выполняющий загрузку и проверку модулей ядра Linux на узле для работы в режиме RPC-with-TLS;
- **net-handshake-checker** — init-контейнер, выполняющий однократный запуск `tshd` для проверки работоспособности процедуры TLS handshake на уровне ядра ОС;
- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно спецификации CSI;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-

сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC `CreateVolume` или `DeleteVolume`. Также использует RPC `ValidateVolumeCapabilities` для проверки совместимости;
 - **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC `ControllerPublishVolume` и `ControllerUnpublishVolume`;
 - **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC `ControllerExpandVolume`, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability `EXPAND_VOLUME`;
 - **snapshotter** (external-snapshotter) — работает совместно с модулем `snapshot-controller`, следит за ресурсами VolumeSnapshotContent, а также управляет снимками томов через RPC `CreateSnapshot`, `DeleteSnapshot` и `ListSnapshots` (если драйвер это поддерживает);
 - **livenessprobe** — отслеживает состояние CSI-драйвера через RPC `Probe` из Identity Service и предоставляет HTTP-эндпоинт `/healthz`, за которым следит kubelet. При неуспешной `livenessProbe` kubelet перезапускает под csi-controller.
2. **Csi-node** (DaemonSet) — Node Plugin, работающий на узлах кластера с лейблом `storage.deckhouse.io/csi-nfs-node` и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability `CAP_SYS_ADMIN`. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **wait-rpcbind** — init-контейнер, ожидающий доступности Unix-сокета службы `rpcbind`;
- **ktls-enabler** — init-контейнер, выполняющий загрузку и проверку модулей ядра Linux на узле для работы в режиме RPC-with-TLS;
- **net-handshake-checker** — init-контейнер, выполняющий однократный запуск `tlshd` для проверки работоспособности процедуры TLS handshake на уровне ядра ОС;
- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **tlshd** — сайдкар-контейнер, обеспечивающий работу в режиме RPC-with-TLS;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC `GetPluginInfo` и `NodeGetInfo`, чтобы получить информацию о плагине и узле. Взаимодействуют с контейнером **node** по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment и VolumeSnapshotContent.
2. **Сетевое файловое хранилище NFS**:
 - подключение и отключение NFS-томов от узлов;
 - управление снимками.

С драйвером взаимодействуют следующие внешние компоненты:

- Kubelet:
 - проверяет `livenessProbe` CSI-драйвера;
 - регистрирует Node Plugin;
 - вызывает RPC `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume` и `NodeExpandVolume` в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокеты.

Дополнительные ресурсы

- Документация модуля `snapshot-controller`

CSI-драйвер S3

CSI-драйвер `csi-s3` — это реализация стандарта Container Storage Interface (CSI) для обеспечения работы с томами на основе S3-хранилищ в Deckhouse Kubernetes Platform (DKP).

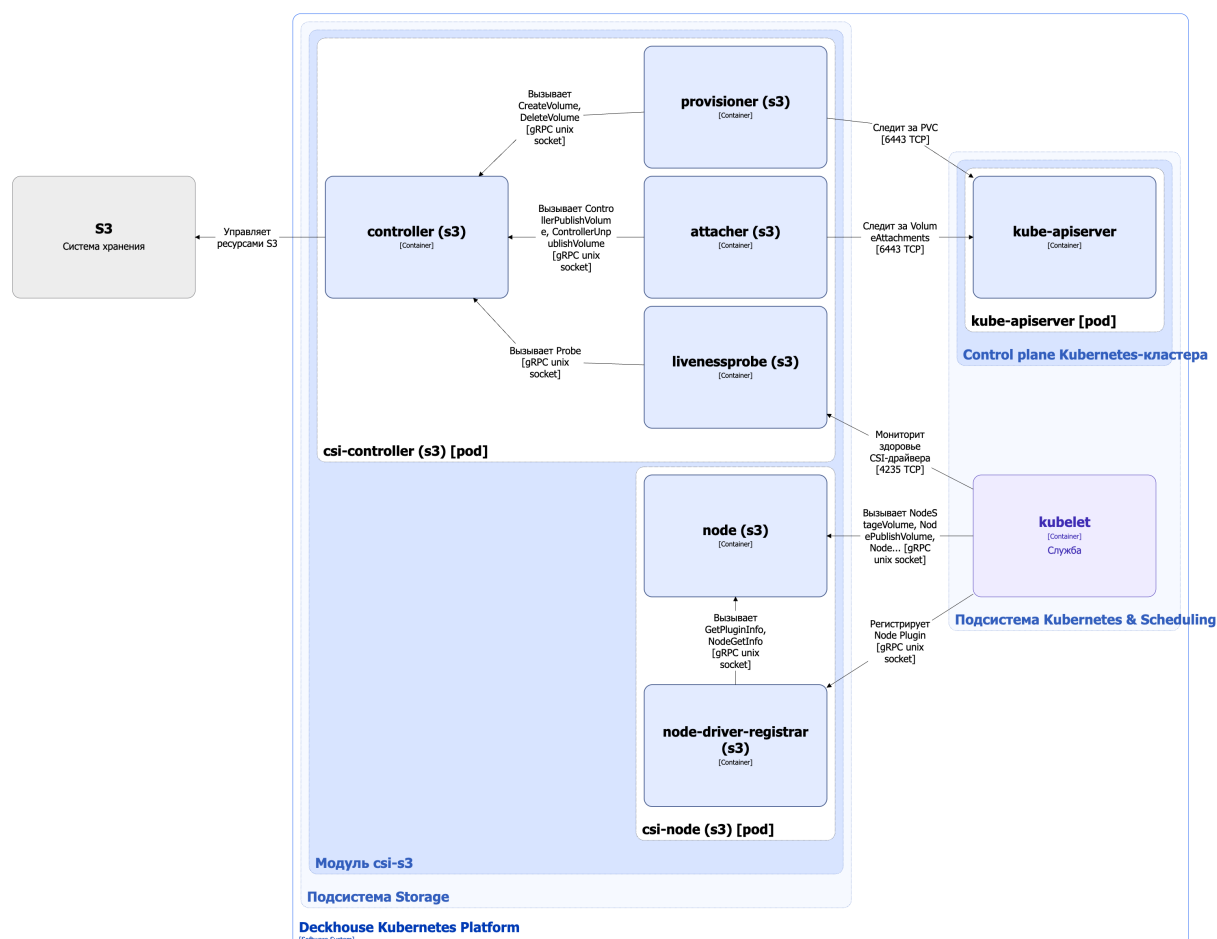
Архитектура CSI-драйвера (csi-s3)

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера (`csi-s3`) на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер `csi-s3` состоит из следующих компонентов:

- Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов.

Состоит из следующих контейнеров:

- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно спецификации CSI;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC CreateVolume или DeleteVolume. Также использует RPC ValidateVolumeCapabilities для проверки совместимости;
- **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC ControllerPublishVolume и ControllerUnpublishVolume;
- **livenessprobe** — отслеживает состояние CSI-драйвера через RPC Probe из Identity Service и предоставляет HTTP-эндпоинт /healthz, за которым следит kubelet. При неуспешной livenessProbe kubelet перезапускает под csi-controller.

2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability CAP_SYS_ADMIN. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC GetPluginInfo и NodeGetInfo, чтобы получить информацию о плагине и узле. Взаимодействует с контейнером **node** по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment.
2. **S3-хранилище** — создание и удаление томов, подключение и отключение томов от узлов.

С драйвером взаимодействуют следующие внешние компоненты:

- Kubelet:
 - проверяет livenessProbe CSI-драйвера;
 - регистрирует Node Plugin;
 - вызывает RPC NodeStageVolume, NodeUnstageVolume, NodePublishVolume и NodeUnpublishVolume в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокет.

CSI-драйвер SCSI Generic

CSI-драйвер `csi-scsi-generic` — это реализация стандарта Container Storage Interface (CSI) для работы с томами на СХД с подключением по SCSI в Deckhouse Kubernetes Platform (DKP). CSI-драйвер поддерживает технологии подключения iSCSI и Fibre Channel (FC).

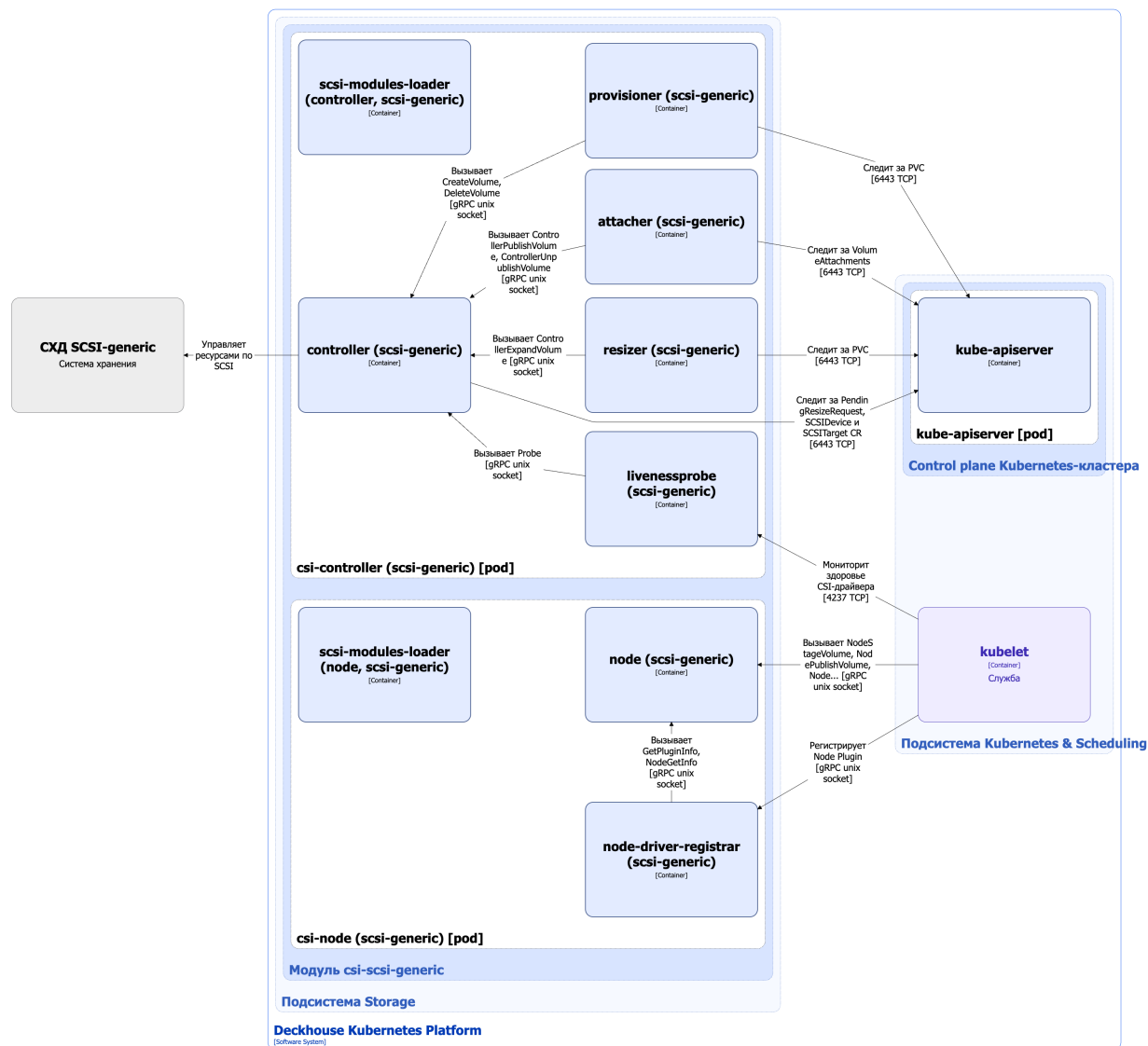
Архитектура CSI-драйвера (csi-scsi-generic)

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера (csi-scsi-generic) на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер `csi-scsi-generic` состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов.

Состоит из следующих контейнеров:

- **scsi-modules-loader** — init-контейнер, обеспечивающий загрузку модуля ядра Linux для работы с iSCSI (`iscsi_tcp`);
- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно

спецификации CSI;

- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC CreateVolume или DeleteVolume;
- **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел;
- **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC ControllerExpandVolume, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability EXPAND_VOLUME. Контроллер не увеличивает размер логических томов хранения данных (Logical Unit Number, LUN) на СХД, а обновляет информацию в метаданных кластера;
- **livenessprobe** — отслеживает состояние CSI-драйвера через RPC Probe из Identity Service и предоставляет HTTP-эндпоинт /healthz, за которым следит kubelet. При неуспешной livenessProbe kubelet перезапускает под csi-controller.

2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.



У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability CAP_SYS_ADMIN. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **scsi-modules-loader** — init-контейнер, обеспечивающий загрузку модуля ядра для работы с iSCSI (iscsi_tcp);
- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC GetPluginInfo и NodeGetInfo, чтобы получить информацию о плагине и узле. Взаимодействует с контейнером node по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolume, PersistentVolumeClaim, VolumeAttachment, PendingResizeRequest, SCSIDevice и SCSTarget.
2. **СХД с подключением по SCSI** — координация использования уже доступных SCSI-устройств, их привязка/очистка и подключение на узлах.

С драйвером взаимодействуют следующие внешние компоненты:

- Kubelet:
 - проверяет livenessProbe CSI-драйвера;
 - регистрирует Node Plugin;
 - вызывает RPC NodeStageVolume, NodeUnstageVolume, NodePublishVolume и NodeUnpublishVolume в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокет.

CSI-драйвер VMware vSphere

Для управления постоянными томами хранения в Deckhouse Kubernetes Platform (DKP) используется CSI-драйвер (плагин).

Container Storage Interface (CSI) — это стандартный интерфейс, который унифицирует доступ к хранилищам и упрощает интеграцию различных систем хранения в кластеры.

В модуле `cloud-provider-vsphere` используется `csi-vsphere` (основан на vSphere CSI Driver), который отличается от драйверов, применяемых в DKP, наличием компонента `syncer`, специфичного для VMware vSphere.

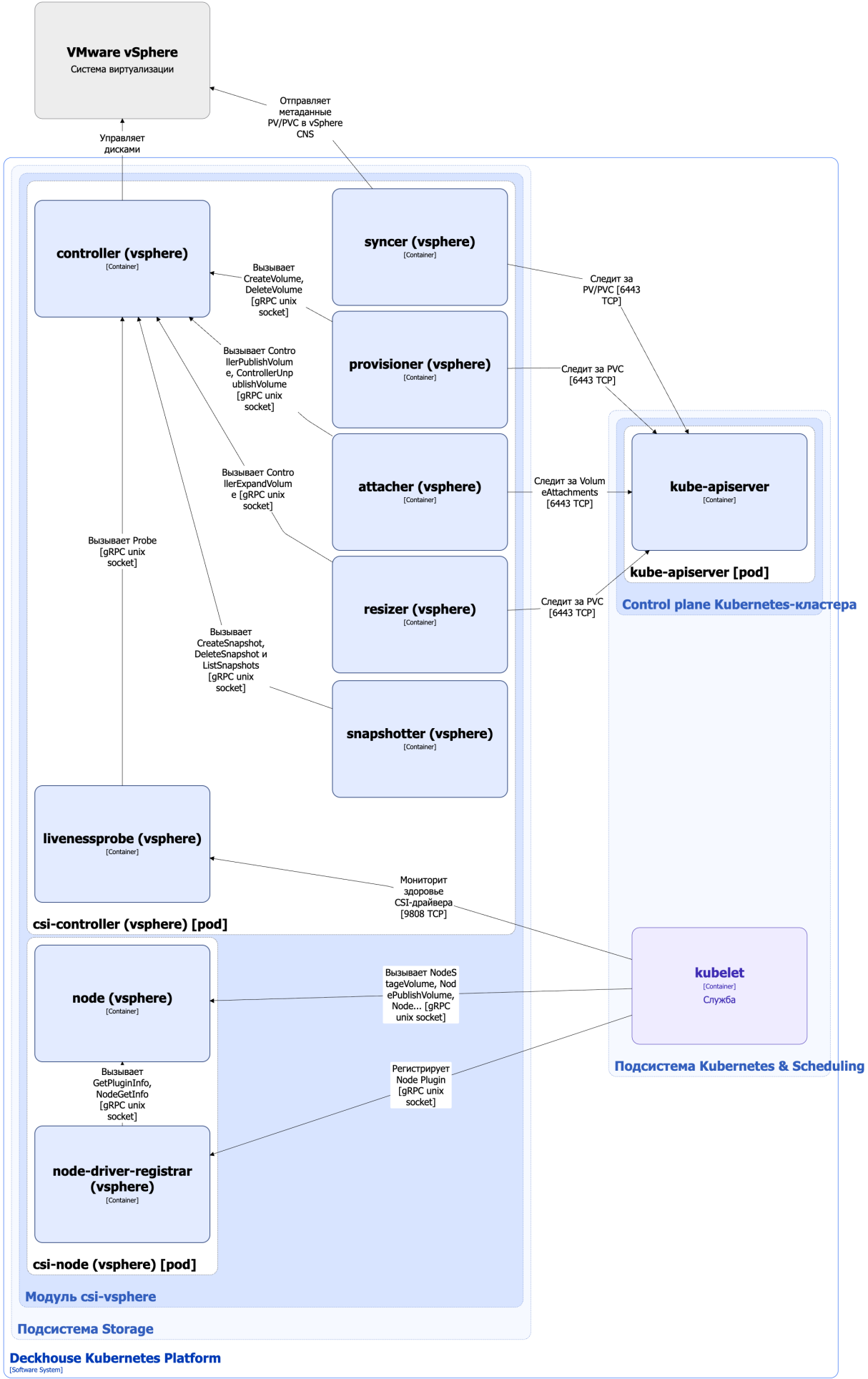
Архитектура драйвера

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура CSI-драйвера `csi-vsphere` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты драйвера

CSI-драйвер состоит из следующих компонентов:

1. **Csi-controller** (Deployment) — Controller Plugin, отвечающий за глобальные операции с томами: создание и удаление, подключение и отключение от узлов, а также управление снимками.

Состоит из следующих контейнеров:

- **controller** — основной контейнер, реализующий функциональность CSI-драйвера (capabilities) в виде gRPC-сервисов Identity Service и Controller Service согласно спецификации CSI;
- **сайдкар-контейнеры контроллера** — поддерживаемые сообществом Kubernetes внешние контроллеры (external controllers).

Они необходимы, поскольку persistent volume controller, запущенный в kube-controller-manager (компонент control plane кластера DKP), не имеет интерфейса взаимодействия с CSI-драйверами. Внешние контроллеры следят за ресурсами PersistentVolumeClaim и вызывают соответствующие функции CSI-драйвера в контейнере controller. Они также выполняют служебные функции, такие как получение информации о плагине и его capabilities или проверка состояния драйвера (liveness probe).

Внешние контроллеры взаимодействуют с контейнером controller по gRPC через Unix-сокеты.

В csi-controller входят следующие внешние контроллеры:

- **provisioner** (external-provisioner) — отслеживает ресурсы PersistentVolumeClaim и вызывает RPC CreateVolume или DeleteVolume. Также использует RPC ValidateVolumeCapabilities для проверки совместимости;
 - **attacher** (external-attacher) — отслеживает ресурсы VolumeAttachment после того, как под запланирован на узел, а также подключает и отключает тома через RPC ControllerPublishVolume и ControllerUnpublishVolume;
 - **resizer** (external-resizer) — отслеживает обновления ресурсов PersistentVolumeClaim, расширяет тома с помощью RPC ControllerExpandVolume, если пользователь запросил больше дискового пространства для PVC и драйвер поддерживает capability EXPAND_VOLUME;
 - **snapshotter** (external-snapshotter) — работает совместно с модулем snapshot-controller, следит за ресурсами VolumeSnapshotContent, а также управляет снимками томов через RPC CreateSnapshot, DeleteSnapshot и ListSnapshots (если драйвер это поддерживает);
 - **livenessprobe** — отслеживает состояние CSI-драйвера через RPC Probe из Identity Service и предоставляет HTTP-эндпоинт /healthz, за которым следит kubelet. При неуспешной livenessProbe kubelet перезапускает под csi-controller.
 - **syncer** — специфичный для VMware vSphere компонент, который синхронизирует метаданные PersistentVolumes, PersistentVolumeClaims и Pods с данными в компоненте VMware vSphere Cloud Native Storage (CNS). CNS — это control plane, который управляет томами различных типов в системе хранения VMware vSphere.
2. **Csi-node** (DaemonSet) — Node Plugin, работающий на всех узлах кластера и отвечающий за локальное монтирование и размонтирование томов.

i

Внимание. У плагина есть привилегированный доступ к файловой системе каждого узла. В Linux для этого требуется capability CAP_SYS_ADMIN. Это необходимо для выполнения операций монтирования и работы с блочными устройствами.

Состоит из следующих контейнеров:

- **node** — основной контейнер, реализующий функции CSI-драйвера в виде gRPC-сервисов Identity Service и Node Service согласно спецификации CSI;
- **node-driver-registrar** — сайдкар-контейнер, регистрирующий Node Plugin в kubelet. Вызывает в контейнере node RPC GetPluginInfo и NodeGetInfo, чтобы получить информацию о плагине и узле. Взаимодействуют с контейнером **node** по gRPC через Unix-сокет.

Взаимодействия драйвера

Драйвер взаимодействует со следующими компонентами:

1. **Kube-apiserver** — мониторинг ресурсов PersistentVolumeClaim, VolumeAttachment и VolumeSnapshotContent.
2. **VMware vSphere**:
 - создание и удаление томов;
 - подключение и отключение томов от узлов;
 - управление снимками;
 - отправка и синхронизация метаданных PersistentVolumes и PersistentVolumeClaims в CNS.

С драйвером взаимодействуют следующие внешние компоненты:

1. Kubelet:
 - проверяет livenessProbe CSI-драйвера;
 - регистрирует Node Plugin;
 - вызывает RPC NodeStageVolume, NodeUnstageVolume, NodePublishVolume, NodeUnpublishVolume и NodeExpandVolume в Node Plugin.

Kubelet взаимодействует с Node Plugin по gRPC через Unix-сокеты.

Дополнительные ресурсы

- Документация модуля `cloud-provider-vsphere`
- Документация модуля `csi-vsphere`
- Документация модуля `snapshot-controller`

Подсистема Observability

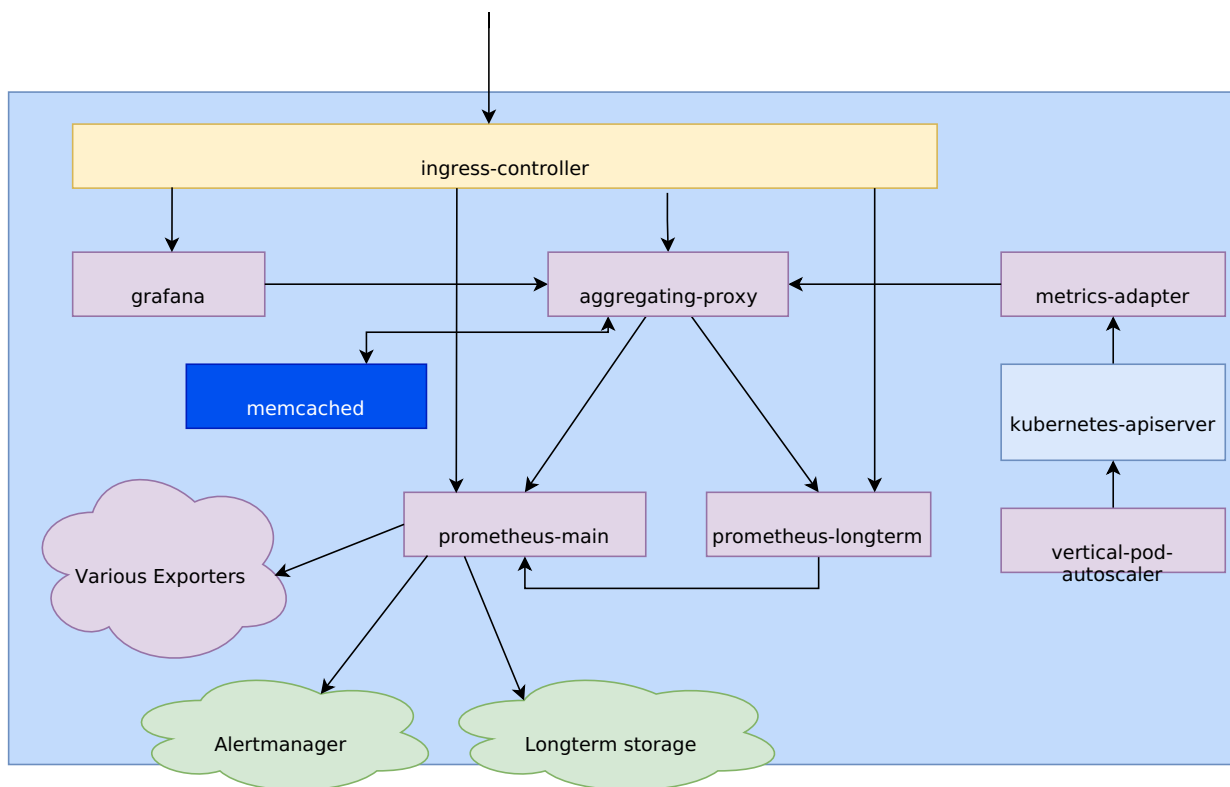
В данном подразделе описывается архитектура подсистемы Observability (подсистемы наблюдаемости) Deckhouse Kubernetes Platform (DKP).

В подсистему Observability входят следующие модули:

- `prometheus` — разворачивает стек мониторинга с предустановленными параметрами для DKP и приложений, что упрощает начальную настройку;
- `operator-prometheus` — устанавливает Prometheus Operator, который автоматизирует развёртывание и управление инстансами Prometheus;
- `prometheus-metrics-adapter` — позволяет автоскейлерам HPA и VPA использовать метрики мониторинга для принятия решений о масштабировании;
- `log-shipper` — упрощает настройку сбора логов в Kubernetes-кластере;
- `loki` — разворачивает в кластере хранилище оперативных логов на базе Grafana Loki;
- `observability` — расширяет функциональность модуля `prometheus` и веб-интерфейса Deckhouse, предоставляя дополнительные возможности для гибкого управления метриками, дашбордами и алертами, а также средства разграничения доступа к ним;
- `extended-monitoring` — расширяет возможности мониторинга кластера за счёт дополнительных Prometheus-экспортеров, которые позволяют выявлять потенциальные проблемы до того, как они скажутся на работе сервисов;
- `monitoring-custom` — упрощает настройку мониторинга пользовательских приложений, требуя только указания определенного лейбла для нужного приложения;
- `monitoring-deckhouse` — обеспечивает мониторинг компонентов и сервисов DKP;
- `monitoring-kubernetes` — обеспечивает прозрачный и своевременный контроль состояния всех узлов кластера и ключевых инфраструктурных компонентов;
- `monitoring-kubernetes-control-plane` — организует безопасный сбор метрик и предоставляет базовый набор правил мониторинга компонентов control plane кластера;
- `upmeter` — проверяет доступность платформы и состояние компонентов кластера в реальном времени и выводит информацию на соответствующие дашборды.

Архитектура мониторинга в Deckhouse Kubernetes Platform

Состав и схема взаимодействия компонентов мониторинга



Компоненты, устанавливаемые DKP

Компонент	Описание
prometheus-operator	Модуль DKP, отвечающий за запуск Prometheus в кластере.
prometheus-main	Основной Prometheus, который выполняет scrape каждые 30 секунд (с помощью параметра <code>scrapeInterval</code> можно изменить это значение). Он обрабатывает все правила, отправляет алерты и является основным источником данных.
prometheus-longterm	Дополнительный Prometheus, хранящий выборку разреженных данных из основного prometheus-main
aggregating-proxy	Агрегирующий и кеширующий прокси, объединяющий main и longterm в один источник. Помогает избежать провалов в данных при недоступности одного из Prometheus.
memcached	Сервис кеширования данных в оперативной памяти.
grafana	UI для отображения метрик в формате дашбордов.
metrics-adapter	Компонент, предоставляющий API Kubernetes для доступа к метрикам. Необходим для правильной работы VPA.
Различные exporter'ы	Набор готовых exporter'ов Prometheus для всех необходимых метрик: <code>kube-state-metrics</code> , <code>node-exporter</code> , <code>oomkill-exporter</code> , <code>image-availability-exporter</code> .
upmeter	Модуль для оценки доступности компонентов DKP.
trickster	Кеширующий прокси, снижающий нагрузку на Prometheus. В ближайшем времени будет deprecated.

Prometheus

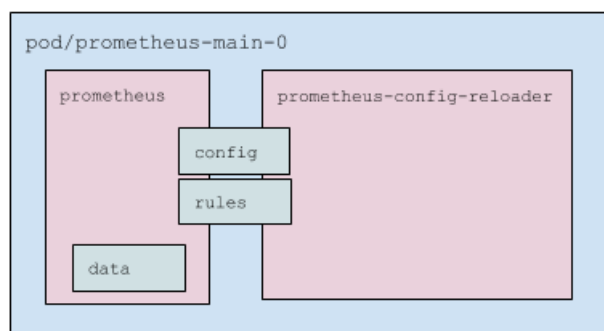
Prometheus собирает метрики и выполняет правила:

- Для каждого *target* (цели мониторинга) с заданной периодичностью `scrape_interval` Prometheus выполняет HTTP-запрос на этот *target*, получает в ответ метрики в собственном формате и сохраняет их в свою базу данных.
- Каждый `evaluation_interval` обрабатывает правила (*rules*), на основании чего:
 - отправляет алерты;
 - или сохраняет новые метрики (результат выполнения правил) в свою базу данных.

Prometheus устанавливается модулем `prometheus-operator` DKP, который выполняет следующие функции:

- определяет следующие кастомные ресурсы:
 - `Prometheus` — определяет инсталляцию (кластер) *Prometheus*;
 - `ServiceMonitor` — определяет, как собирать метрики с сервисов;
 - `Alertmanager` — определяет кластер *Alertmanager*’ов;
 - `PrometheusRule` — определяет список *Prometheus rules*.
- следит за этими ресурсами, а также:
 - генерирует `StatefulSet` с самим *Prometheus*;
 - создает секреты с необходимыми для работы Prometheus конфигурационными файлами (`prometheus.yaml` — конфигурация Prometheus, и `configmaps.json` — конфигурация для `prometheus-config-reloader`);
 - следит за ресурсами `ServiceMonitor` и `PrometheusRule` и на их основании обновляет конфигурационные файлы *Prometheus* через внесение изменений в секреты.

Что в поде с Prometheus?



- Два контейнера:
 - `prometheus` — сам Prometheus;
 - `prometheus-config-reloader` — обвязка, которая:
 - следит за изменениями `prometheus.yaml` и, при необходимости, вызывает `reload` конфигурации Prometheus’у (специальным HTTP-запросом, см. подробнее ниже);
 - следит за `PrometheusRule`’ами (см. подробнее ниже) и по необходимости скачивает их и перезапускает Prometheus.
- Pod использует три volume:
 - `config` — примонтированный secret (два файла: `prometheus.yaml` и `configmaps.json`). Подключен в оба контейнера;
 - `rules` — `emptyDir`, который наполняет `prometheus-config-reloader`, а читает `prometheus`. Подключен в оба контейнера, но в `prometheus` в режиме `read only`;
 - `data` — данные Prometheus. Подмонтирован только в `prometheus`.

Как настраивается Prometheus?

- У сервера Prometheus есть *config* и есть *rule files* (файлы с правилами);
- В *config* имеются следующие секции:
 - `scrape_configs` — настройки поиска *target’ов* (целей для мониторинга, см. подробней следующий раздел);
 - `rule_files` — список директорий, в которых лежат *rule’ы*, которые необходимо загружать:

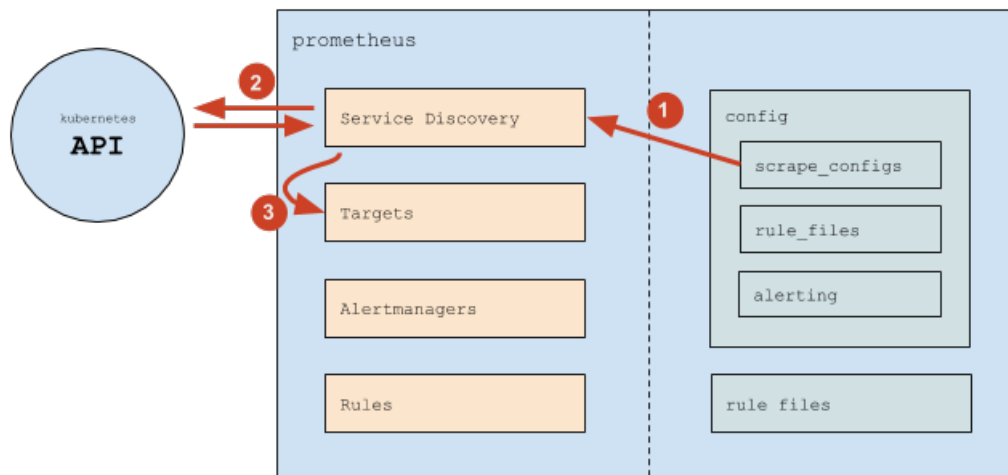
```
rule_files:
- /etc/prometheus/rules/rules-0/*
- /etc/prometheus/rules/rules-1/*
```

- `alerting` — настройки поиска *Alert Manager’ов*, в которые слать алерты. Секция очень

похожа на `scrape_configs`, только результатом ее работы является список *endpoint'ов*, в которые Prometheus будет слать алерты.

Где Prometheus берет список *target'ов*?

- В целом Prometheus работает следующим образом:



- (1) Prometheus читает секцию конфига `scrape_configs`, согласно которой настраивает свой внутренний механизм `Service Discovery`;
 - (2) Механизм `Service Discovery` взаимодействует с API Kubernetes (в основном — получает *endpoint'ы*);
 - (3) На основании происходящего в Kubernetes механизм `Service Discovery` обновляет `Targets` (список *target'ов*).
- В `scrape_configs` указан список *scrape job'ов* (внутреннее понятие Prometheus), каждый из которых определяется следующим образом:

```
scrape_configs:
  # Общие настройки
  - job_name: d8-monitoring/custom/0 # просто название scrape job'a,
    # показывается в разделе Service Discovery
    scrape_interval: 30s # как часто собирать данные
    scrape_timeout: 10s # таймаут на запрос
    metrics_path: /metrics # path, который запрашивать
    scheme: http # http или https
    # Настройки service discovery
    kubernetes_sd_configs: # означает, что target'ы мы получаем из
      - Kubernetes
        api_server: null # означает, что адрес API-сервера
        # использовать из переменных окружения (которые есть в каждом Pod'e)
        role: endpoints # target'ы брать из endpoint'ов
        namespaces:
          names: # искать endpoint'ы только в этих namespace'ax
            - foo
            - baz
        # Настройки "фильтрации" (какие эндпоинты брать, а какие нет) и "релейблинга"
        # (какие лейблы добавить или удалить, на все получаемые метрики)
        relabel_configs:
          # Фильтр по значению label'a prometheus_custom_target (полученного из
          # связанного с endpoint'ом service'a)
          - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
            regex: .+ # подходит любой НЕ пустой лейбл
            action: keep
          # Фильтр по имени порта
          - source_labels: [__meta_kubernetes_endpointslice_port_name]
            regex: http-metrics # подходит, только если порт называется
            http-metrics
            action: keep
          # Добавляем label job, используем значение label'a prometheus_custom_target у
```

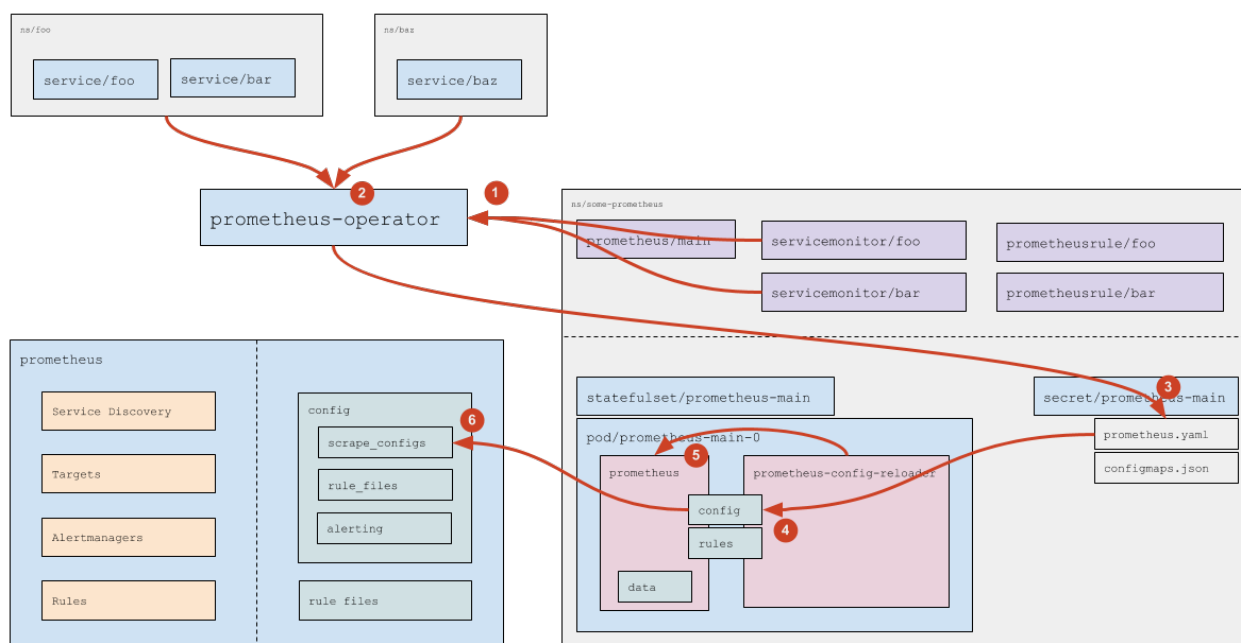
```

service'a, к которому добавляем префикс "custom-"
#
# Лейбл job это служебный лейбл Prometheus:
# * он определяет название группы, в которой будет показываться target на
странице targets
# * и конечно же он будет у каждой метрики, полученной у этих target'ов,
чтобы можно было удобно фильтровать в rule'ax и dashboard'ax
- source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
  regex: (.*)
  target_label: job
  replacement: custom-$1
  action: replace
# Добавляем label namespace
- source_labels: [__meta_kubernetes_namespace]
  regex: (.*)
  target_label: namespace
  replacement: $1
  action: replace
# Добавляем label service
- source_labels: [__meta_kubernetes_service_name]
  regex: (.*)
  target_label: service
  replacement: $1
  action: replace
# Добавляем label instance (в котором будет имя Pod'a)
- source_labels: [__meta_kubernetes_pod_name]
  regex: (.*)
  target_label: instance
  replacement: $1
  action: replace

```

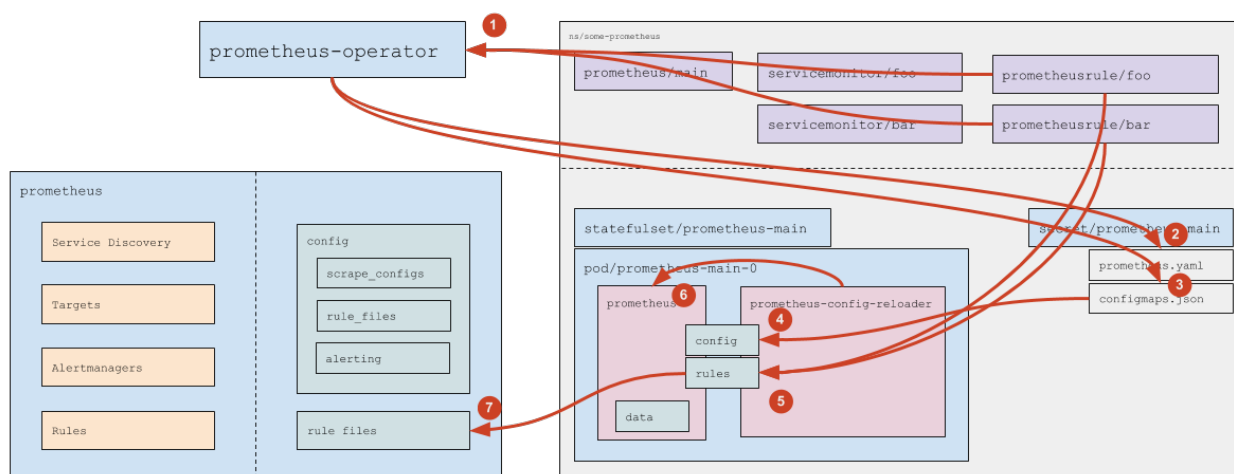
- Таким образом, Prometheus сам отслеживает:
 - добавление и удаление Pod'ов (при добавлении/удалении Pod'ов Kubernetes изменяет endpoint'ы, а Prometheus это видит и добавляет/удаляет *target'ы*);
 - добавление и удаление сервисов (точнее endpoint'ов) в указанных пространствах имён;
- Изменение конфига требуется в следующих случаях:
 - нужно добавить новый scrape config (обычно — новый вид сервисов, которые надо мониторить);
 - нужно изменить список пространств имён.

Как обрабатываются Service Monitor'ы?



1. Prometheus Operator читает (а также следит за добавлением/удалением/изменением) Service Monitor'ы (какие именно Service Monitor'ы — указано в самом ресурсе `prometheus`, см. подробней официальную документацию).
2. Для каждого Service Monitor'a, если в нем НЕ указан конкретный список namespace'ов (указано `any: true`), Prometheus Operator вычисляет (обращаясь к API Kubernetes) список namespace'ов, в которых есть Service'ы (подходящие под указанные в Service Monitor'e label'ы).
3. На основании прочитанных ресурсов `servicemonitor` (см. официальную документацию) и на основании вычисленных namespace'ов Prometheus Operator генерирует часть конфига (секцию `scrape_configs`) и сохраняет конфиг в соответствующий Secret.
4. Штатными средствами самого Kubernetes данные из секрета прилетают в Pod (файл `prometheus.yaml` обновляется).
5. Изменение файла замечает `prometheus-config-reloader`, который по HTTP отправляет запрос Prometheus'у на перезагрузку.
6. Prometheus перечитывает конфиг и видит изменения в `scrape_configs`, которые обрабатывает уже согласно своей логике работы (см. подробнее выше).

Как обрабатываются кастомные ресурсы с *rule'ами*?



1. Prometheus Operator следит за PrometheusRule'ами (подходящими под указанный в ресурсе `prometheus ruleSelector`).
2. Если появился новый (или был удален существующий) PrometheusRule — Prometheus Operator обновляет `prometheus.yaml` (а дальше срабатывает логика в точности соответствующая обработке Service Monitor'ов, которая описана выше).
3. Как в случае добавления/удаления PrometheusRule'a, так и при изменении содержимого PrometheusRule'a, Prometheus Operator обновляет ConfigMap `prometheus-main-rulefiles-0`.
4. Штатными средствами самого Kubernetes данные из ConfigMap прилетают в Pod
5. Изменение файла замечает `prometheus-config-reloader`, который:
 - скачивает изменившиеся ConfigMap'ы в директорию `rules` (это `emptyDir`)
 - по HTTP отправляет запрос Prometheus'у на перезагрузку
6. Prometheus перечитывает конфиг и видит изменившиеся *rule'ы*.

Архитектура оценки доступности компонентов DKP (upmeter)

Оценка доступности в DKP осуществляется модулем `upmeter`.

Состав модуля `upmeter`:

- **agent** — работает на master-узлах и делает пробы доступности, отправляет результаты на сервер.
- **upmeter** — собирает результаты и поддерживает API-сервер для их извлечения.
- **front**:
 - **status** — показывает уровень доступности за последние 10 минут (требуется авторизация, но ее можно отключить);
 - **webui** — показывает дашборд со статистикой по пробам и группам доступности (требуется авторизация).
- **smoke-mini** — поддерживает постоянное *smoke-тестирование* с помощью StatefulSet.

Модуль отправляет около 100 показаний метрик каждые 5 минут. Это значение зависит от количества включенных модулей Deckhouse Kubernetes Platform.

Дополнительные ресурсы

- Документация модуля `upmeter`

Модуль prometheus

Модуль `prometheus` разворачивает стек мониторинга с предустановленными параметрами для Deckhouse Kubernetes Platform (DKP) и обеспечивает сбор, хранение и обработку метрик кластера и приложений.

Подробнее с описанием модуля можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `prometheus` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Prometheus-main** (StatefulSet) — основной экземпляр Prometheus. Prometheus — это система мониторинга и оповещения, использующая базу данных временных рядов (TSDB или time series database). Она в реальном времени собирает и анализирует метрики работы приложений и серверов. Prometheus-main собирает метрики с настроенных объектов мониторинга каждые 30 секунд. С помощью параметра `scrapeInterval` можно изменить это значение.

В `prometheus-main` может использоваться оригинальный («vanilla») Prometheus или Deckhouse Prom++ — высокопроизводительный форк Prometheus с открытым исходным кодом, разработанный для сокращения потребления памяти при сохранении полной совместимости с оригинальным проектом. В модуле по умолчанию используется Deckhouse Prom++. Есть возможность переключиться с Deckhouse Prom++ на оригинальный Prometheus. В этом случае потребуются миграция данных журнала упреждающей записи (WAL или write-ahead log), поскольку в Deckhouse Prom++ используется свой формат журнала WAL. Миграция осуществляется автоматически при помощи `init`-контейнера `promptool`.

Prometheus-main является основным источником данных. Он собирает метрики, обрабатывает настроенные правила и отправляет алерты в соответствии с заданной конфигурацией. Инсталляцию Prometheus, а также его конфигурацию создаёт Prometheus Operator на основании следующих кастомных ресурсов:

- Prometheus — описывает инсталляцию (кластер) Prometheus;
- ServiceMonitor — задаёт настройки сбора метрик с набора сервисов;
- PrometheusRule — содержит набор правил Prometheus.

Prometheus Operator отслеживает ресурсы Prometheus и для каждого генерирует:

- StatefulSet с самим Prometheus;
- секрет `prometheus-main` с `prometheus.yaml` (основной конфигурационный файл) и `configmaps.json` (конфигурационный файл для контейнера `config-reloader`, описанного ниже). Секрет `prometheus-main` монтируется в под `prometheus-main` и используется контейнером `config-reloader`.

Prometheus Operator отслеживает ресурсы ServiceMonitor и PrometheusRule и на их основании обновляет конфигурацию (`prometheus.yaml` и `configmaps.json`) в указанном выше секрете.

Подробнее с описанием работы Prometheus Operator можно ознакомиться в разделе документации модуля `operator-prometheus`.

Подробнее с описанием работы компонента `prometheus-main` можно ознакомиться в разделе «Архитектура мониторинга».

Prometheus-main состоит из следующих контейнеров:

- **init-config-reloader** — init-контейнер, выполняющий однократный запуск config-reloader для загрузки конфигурации Prometheus;
- **prompptestool** — init-контейнер, выполняющий автоматическую миграцию данных журнала WAL в случае переключения с Deckhouse Prom++ на оригинальный Prometheus и наоборот;
- **config-reloader** — сайдкар-контейнер, который следит за изменениями в файле конфигурации `prometheus.yaml` и, при необходимости, вызывает перезагрузку конфигурации Prometheus (HTTP-запросом на специальный эндпоинт `/-/reload`). Config-reloader является утилитой из Open Source-проекта Prometheus Operator;
- **prometheus** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к серверу Prometheus. Является Open Source-проектом.

2. **Prometheus-longterm** (StatefulSet) — дополнительный экземпляр Prometheus, хранящий выборку разреженных метрик из основного Prometheus (prometheus-main). Это позволяет пользователям просматривать и анализировать исторические тренды за длительный период времени. Prometheus-longterm получает данные благодаря настроенному механизму федерации с основным экземпляром Prometheus.

В prometheus-longterm также может использоваться оригинальный Prometheus или Deckhouse Prom++. Состав контейнеров и принцип их работы у prometheus-longterm совпадает с prometheus-main.

3. **Grafana-v10** — необязательный компонент Grafana, предоставляющий веб-интерфейс для визуализации данных мониторинга. Grafana отображает дашборды, поставляемые вместе с модулями DKP. Grafana умеет работать в режиме высокой доступности, не хранит состояние и настраивается с помощью кастомных ресурсов. Компонент включён по умолчанию, но при желании его можно отключить с помощью параметра `settings.grafana.enabled`.



Grafana-v10 будет отключена в будущих релизах DKP. Для просмотра дашбордов мониторинга используйте веб-интерфейс Deckhouse.

Состоит из следующих контейнеров:

- **dashboard-provisioner** — сайдкар-контейнер, который следит за кастомными ресурсами `GrafanaDashboardDefinition` и, при появлении нового ресурса, добавляет описанные в них дашборды в папку Grafana;
- **grafana** — основной контейнер. Является Open Source-проектом;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к серверу Grafana и его метрикам. Подробно описан выше.

4. **Aggregating-proxy** — выполняет кэширование метрик, сбор данных с нескольких экземпляров Prometheus (если они работают в режиме высокой доступности), дедупликацию данных и вычисление запросов.

Состоит из следующих контейнеров:

- **wait-memcached** — init-контейнер, ожидающий доступности компонента memcached по сети. Aggregating-proxy использует memcached для кэширования метрик в оперативной памяти;
- **mimir** — сайдкар-контейнер, работающий с компонентом memcached для оптимизации запросов и кэширования данных. При отсутствии данных в кеше, mimir пересылает запрос на компонент prometheus-main через еще один сайдкар-контейнер promxy. Является Open Source-проектом;
- **promxy** — сайдкар-контейнер, проксирующий запросы к компоненту prometheus-main и предоставляющий единый эндпоинт для доступа к данным нескольких экземпляров Prometheus. Является Open Source-проектом;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к контейнеру mimir (запросы к серверу Prometheus и метрикам контейнера) и promxy (запросы к метрикам контейнера). Подробно описан выше.

5. **Memcached** (StatefulSet) — компонент, используемый aggregating-proxy для кэширования метрик Prometheus. Memcached — это программное обеспечение, реализующее сервис кэширования данных в оперативной памяти для ускорения выполнения запросов к метрикам Prometheus.

Состоит из следующих контейнеров:

- **memcached** — основной контейнер. Является Open Source-проектом;

- **exporter** — сайдкар-контейнер, экспортирующий метрики контейнера memcached. Exporter собирает метрики контейнера memcached через сетевое подключение, а также из PID-файла процесса memcached. Является Open Source-проектом.
6. **Trickster** — кеширующий прокси-сервер, снижающий нагрузку на Prometheus. Используется для кеширования и проксирования запросов к prometheus-longterm. В будущих релизах DKP ожидается, что компонент будет признан устаревшим и перестанет поддерживаться.

Состоит из следующих контейнеров:

- **trickster** — основной контейнер. Является Open Source-проектом;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к прокси-серверу и его метрикам. Подробно описан выше.



В будущих релизах DKP alerts-receiver будет удалён из модуля prometheus. Для приема всех алертов будет использоваться компонент Alertmanager модуля observability.

7. **Alerts-receiver** — сервер, совместимый с API Alertmanager. Alerts-receiver принимает базовые алерты от prometheus-main, создаёт на их основе кастомные ресурсы ClusterAlert, обновляет их статусы и удаляет, если алерт больше не активен. Кастомные ресурсы ClusterAlert используются для информирования пользователей DKP об активных алертах и отображаются в веб-интерфейсе Deckhouse. Является разработкой компании «Флант». Состоит из одного контейнера.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- мониторинг кастомных ресурсов PrometheusRule и GrafanaDashboardDefinition;
- управление кастомными ресурсами ClusterAlert;
- авторизация запросов на получение метрик компонентов модуля.

2. Alertmanager — отправка кастомных алертов.

Экземпляр Prometheus, входящий в состав модуля, собирает метрики со всех компонентов DKP:

- компоненты модулей;
- компоненты control plane кластера;
- экспортеры, собирающие метрики загрузки аппаратных ресурсов кластера;
- экспортеры, собирающие метрики ресурсов Kubernetes;
- пользовательские приложения (требуется дополнительная настройка).

Взаимодействия Prometheus с компонентами DKP, связанные со сбором метрик, не показаны на схеме, чтобы не усложнять её большим количеством связей.

С модулем взаимодействуют следующие внешние компоненты:

1. **Ingress-controller** (controller nginx на схеме) — пересылает запросы пользователей к Grafana.

Режим отказоустойчивости и высокой доступности мониторинга (HA)

Модуль prometheus обеспечивает встроенную отказоустойчивость всех своих ключевых компонентов. Все сервисы мониторинга (Prometheus-серверы, системы хранения, прокси и прочие важные компоненты) по умолчанию развёртываются в нескольких копиях. Это гарантирует, что в случае сбоя отдельного экземпляра сервис продолжит работу без потери данных и доступности.

Prometheus — основной компонент сбора метрик — запускается минимум в двух копиях (при наличии достаточного количества узлов в кластере). Все экземпляры Prometheus используют одинаковую конфигурацию и получают одни и те же данные. Чтобы обеспечить бесперебойную работу при отказе одной из копий для обращения к Prometheus используется специальный компонент — aggregating-proxy. Он позволяет объединять метрики нескольких экземпляров Prometheus и всегда возвращать наиболее полные и актуальные данные, даже если одна из копий временно недоступна.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `observability`
- Документация модуля `operator-prometheus`
- Документация модуля `prometheus`

Модуль `observability`

Модуль `observability` расширяет функциональность модуля `prometheus` и веб-интерфейса Deckhouse, предоставляя дополнительные возможности для гибкого управления метриками, дашбордами и алертами, а также средства разграничения доступа к ним.

Возможности модуля:

- **Управление дашбордами** — модуль позволяет пользователям добавлять собственные дашборды в формате Grafana.
- **Управление группами триггеров и метрик** — модуль позволяет создавать и настраивать собственные группы триггеров и метрик.
- **Настройка уведомлений** — модуль позволяет настраивать каналы уведомлений (Telegram, Slack, электронная почта, вебхуки), политики уведомлений, а также дает возможность отключать уведомления при необходимости.
- **Работа с алертами** — модуль предоставляет информацию об активных алертах, а также сохраняет историю завершенных алертов.
- **Предоставление стандартных источников данных.**
- **Поддержка пользовательских источников данных** — модуль дополнительно к предоставляемому набору стандартных источников данных позволяет добавить пользовательские.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `observability` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Observability-controller** — состоит из одного контейнера **observability-controller**, управляет жизненным циклом большинства кастомных ресурсов модуля, таких как `ObservabilityMetricsRulesGroup`, `ObservabilityNotificationChannels`, `ObservabilityNotificationSilence` и других. С полным списком ресурсов, которыми управляет модуль, можно ознакомиться в разделе «Кастомные ресурсы» документации модуля.
2. **Observability-webhook** — состоит из одного контейнера **observability-webhook**, реализующего вебхук-сервер для проверки и изменения кастомных ресурсов через механизмы `Validating/Mutating Admission Controllers`.
3. **Alert-kube-api** — состоит из одного контейнера **alert-kube-api**, реализует Kubernetes Extension API Server, который расширяет Kubernetes API кастомными ресурсами `ObservabilityAlert` и `ClusterObservabilityAlert`. `Alert-kube-api` позволяет запрашивать алерты как кастомные ресурсы, используя в качестве бэкенда компонент `Alertmanager` модуля, и кеширует их в памяти для быстрого доступа.
4. **Alertmanager** — принимает алерты от компонента `prometheus-main` модуля `prometheus`, обрабатывает и отправляет их конечным получателям. DKP поддерживает отправку алертов через следующие каналы доставки:
 - `Email`;

- Telegram;
- Slack;
- Webhook.

Компонент содержит следующие контейнеры:

- **alertmanager** — основной контейнер. В модуле используется в форк оригинального Alertmanager от компании «Флант», поддерживающий мультитенантность: разделение алертов на системные (кластерные) и проектные. Доставка алертов (политики, каналы, сайленсеры) также мультитенантная, что позволяет индивидуально настроить каналы и политики доставки в разных проектах.
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к API-эндпоинту Alertmanager. Является Open Source-проектом.
5. **Grafana** — компонент, предоставляющий веб-интерфейс для визуализации данных мониторинга. В модуле используется форк Grafana от компании «Флант». Используемая модификация Grafana обладает расширенными возможностями, такими как разграничение доступа к метрикам и дашбордам в рамках мультитенантности. Дашборды Grafana модуля observability интегрированы в веб-интерфейс Deckhouse (управление системой мониторинга из одного окна).

Компонент содержит следующие контейнеры:

- **grafana** — основной контейнер;
 - **grafana-kube-storage** — сайдкар-контейнер, реализующий бэкенд для контейнера grafana и предоставляющий управление ресурсами Dashboard и чтение ресурсов Datasource API компонента Grafana. Данные ресурсы позволяют просматривать и управлять дашбордами в пределах неймспейсов (проектов), а также подключать пользовательские источники данных;
 - **nginx** — сайдкар-контейнер, представляющий собой прокси-сервер NGINX, чья основная задача — это раздача статических файлов. Является Open Source-проектом.
6. **Label-enforcer** — компонент, выполняющий авторизацию и проксирование запросов пользователей к источникам метрик (prometheus-main через сервис label-proxy) и логов (loki через сервис logs-gateway), указанным в ресурсах Datasource API компонента grafana. Label-enforcer проверяет RBAC-доступ к данным мониторинга в зависимости от прав пользователя, получает список доступных неймспейсов и обогащает запросы лейблами для возможности фильтрации запрашиваемых данных в рамках неймспейсов пользователей. Подробнее о разграничении доступа к метрикам можно ознакомиться в документации модуля. Label-enforcer обрабатывает запросы на чтение и запись метрик.

Состоит из единственного контейнера **enforcer**.

7. **Opagent** (DaemonSet) — агент, предназначенный для сбора метрик с операционной системы и прикладного программного обеспечения, установленного на серверы. Разработан компанией «Флант» для Deckhouse Observability Platform (DOP) на основе Okagent (агента Okmeter), входящего в состав системы мониторинга Okmeter.

В модуле observability opAgent подключается к managed-сервисам, таким как managed-postgres, managed-memcached, managed-kafka и другим (список поддерживаемых сервисов постоянно расширяется), собирает с них метрики, после чего отправляет их в компонент prometheus-main модуля prometheus. Если включен модуль observability-platform, opAgent собирает метрики с узлов кластера и отправляет их в DOP.

OpAgent отправляет собранные данные по протоколу Prometheus Remote Write:

- метрики managed-сервисов — в Prometheus через label-enforcer;
- метрики с узлов кластера — в DOP.

Состоит из единственного контейнера **opagent**.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - для авторизации запросов к данным мониторинга;
 - для управления кастомными ресурсами модуля.
2. **Prometheus-main** — использует в качестве источника и приемника данных.
3. **Loki** — использует в качестве источника данных.
4. **Получатели алертов** — отправляет алерты.
5. Deckhouse Observability Platform — использует в качестве приемника данных (метрики

узлов).

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver:**

- выполняет проверку и изменение кастомных ресурсов модуля (с помощью validating- и mutating-вебхуков);
- пересылает в alert-kube-api запросы на кастомные ресурсы ObservabilityAlert и ClusterObservabilityAlert.

2. **Prometheus-main** — отправляет алерты в Alertmanager.

3. **Веб-интерфейс Deckhouse** — использует Grafana для визуализации данных мониторинга.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `managed-kafka`
- Документация модуля `managed-memcached`
- Документация модуля `managed-postgres`
- Документация модуля `observability`
- Документация модуля `prometheus`

Модуль extended-monitoring

Модуль `extended-monitoring` расширяет возможности мониторинга кластера за счёт дополнительных экспортеров Prometheus, которые позволяют выявлять потенциальные проблемы до того, как они скажутся на работе сервисов.

Возможности модуля:

- расширенный сбор метрик;
- мониторинг контейнерных образов;
- сбор событий в кластере;
- контроль сертификатов.

Подробнее с настройками и примерами использования модуля можно ознакомиться в разделе документации модуля.

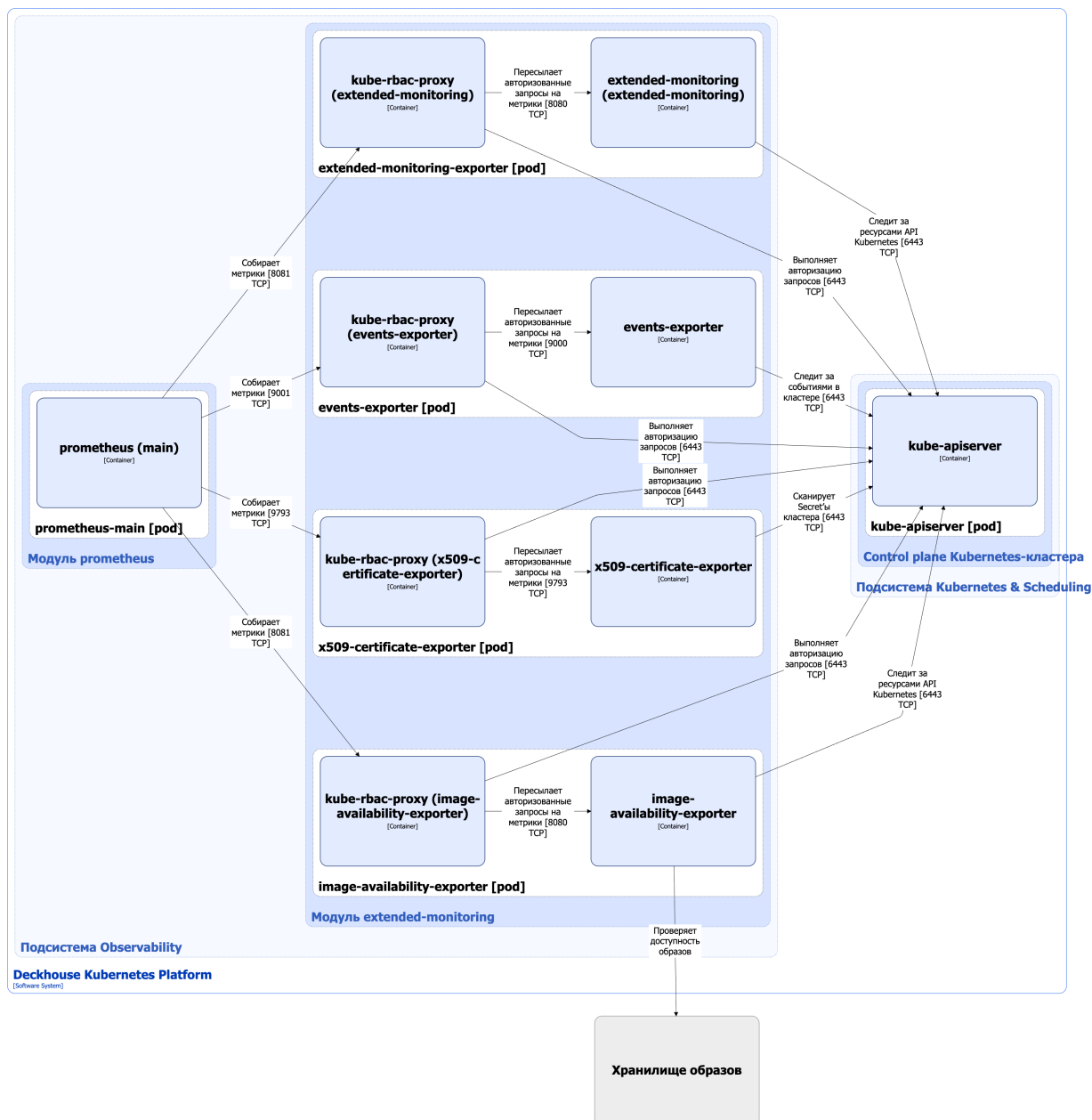
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `extended-monitoring` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Extended-monitoring-exporter** — экспортер Prometheus, собирает и экспортирует метрики по свободному месту и inode на узлах, а также по объектам с лейблом `extended-monitoring.deckhouse.io/enabled=""` в неймспейсе. Подробнее с настройкой `extended-monitoring` метрик можно ознакомиться в разделе документации модуля. В модуле также содержатся Prometheus-правила, на основе которых Prometheus создает алерты. Это позволяет быстрее обнаруживать и диагностировать инциденты.

Состоит из следующих контейнеров:

- **extended-monitoring-exporter** — основной контейнер. Является разработкой компании «Флант»;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам экспортера. Является Open Source-проектом.
2. **Image-availability-exporter** — экспортер Prometheus, который осуществляет мониторинг контейнерных образов запущенных в кластере подов. `Image-availability-exporter` добавляет метрики о недоступности образов в хранилищах образов контейнеров для всех типов рабочей нагрузки (Deployments, StatefulSets, DaemonSets, CronJobs). Модуль также содержит соответствующие Prometheus-правила, на основе которых Prometheus создает алерты. Это помогает заранее узнать о возможных проблемах с запуском или обновлением подов.

Состоит из следующих контейнеров:

- **image-availability-exporter** — основной контейнер. Является разработкой компании «Флант»;
 - **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам экспортера. Подробно описан выше.
3. **Events-exporter** — экспортер Prometheus, который собирает события Kubernetes и отображает их в виде метрик, что позволяет отслеживать динамику изменений и быстрее реагировать на инциденты.

Состоит из следующих контейнеров:

- **events-exporter** — основной контейнер. Является разработкой компании «Флант»;
 - **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам экспортера. Подробно описан выше.
4. **X509-certificate-exporter** — экспортер Prometheus, который обеспечивает контроль сертификатов:
- Сканирует Secret'ы кластера и генерирует метрики об истечении срока действия x509-сертификатов;
 - Модуль также содержит соответствующие Prometheus-правила, на основе которых Prometheus создает алерты. Это позволяет не пропускать критические моменты и вовремя обновлять сертификаты, избегая простоя приложений из-за просроченных сертификатов.

Состоит из следующих контейнеров:

- **x509-certificate-exporter** — основной контейнер. Является Open Source-проектом;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам экспортера. Подробно описан выше.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - следит за ресурсами API Kubernetes;
 - следит за событиями в кластере (ресурсами Events);
 - сканирует Secret'ы кластера;
 - выполняет авторизацию запросов на метрики.
2. **Хранилище образов** — проверяет доступность образов.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — собирает метрики с экспортеров модуля.

Дополнительные ресурсы

- Документация модуля `extended-monitoring`

Модуль monitoring-kubernetes

Модуль `monitoring-kubernetes` обеспечивает прозрачный и своевременный контроль состояния всех узлов кластера и ключевых инфраструктурных компонентов.

Подробнее с возможностями и настройками модуля можно ознакомиться в разделе документации модуля.

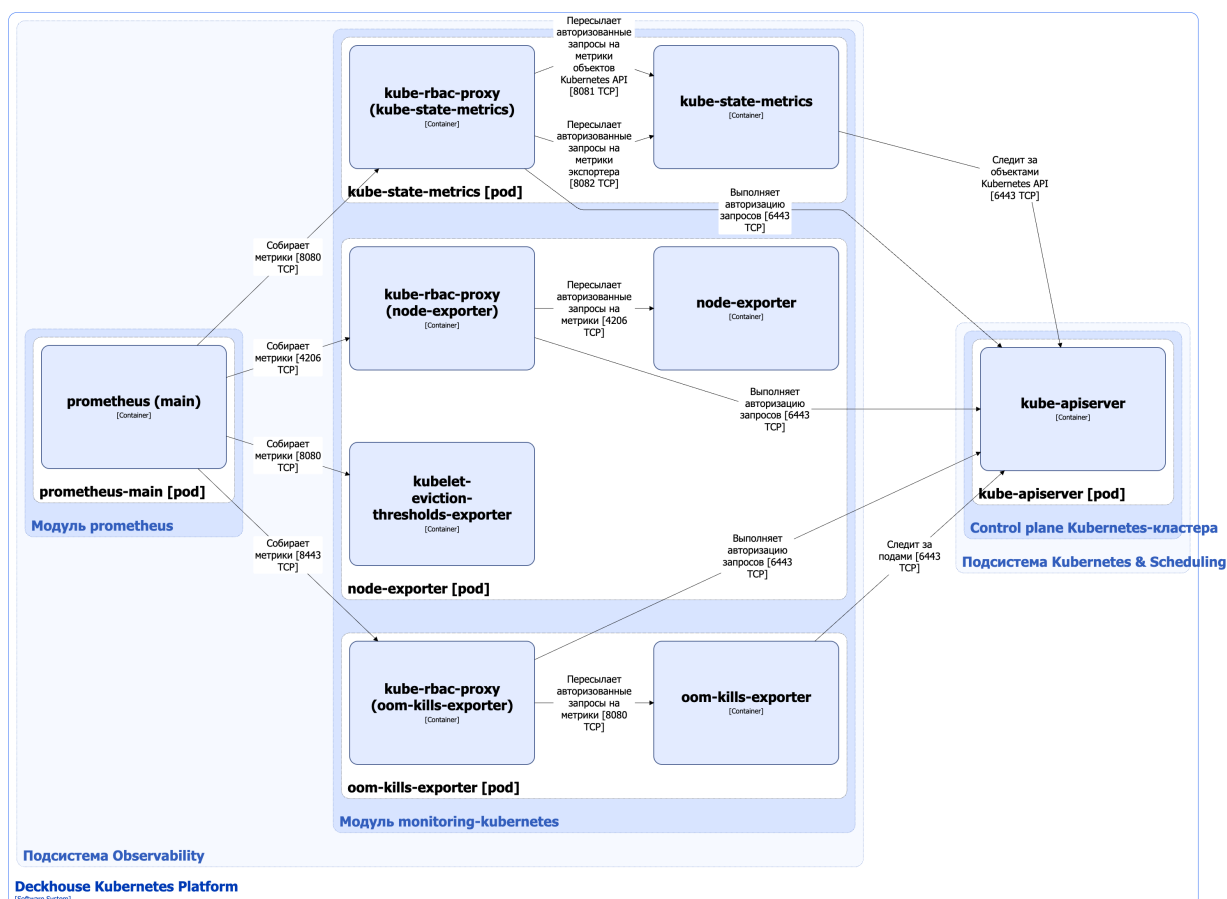
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура модуля `monitoring-kubernetes` на уровне 2 модели C4 и его взаимодействия с другими компонентами Deckhouse Kubernetes Platform (DKP) изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Kube-state-metrics** (Deployment) — экспортер Prometheus, который собирает и предоставляет метрики о состоянии объектов в кластере Kubernetes. Он подключается к API-серверу Kubernetes, анализирует текущее состояние ресурсов (Pods, Services, Deployments, Nodes и других) и генерирует на этой основе метрики.

Состоит из следующих контейнеров:

- **kube-state-metrics** — основной контейнер. Является Open Source-проектом;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам о состоянии объектов Kubernetes API, а также метрикам состояния самого экспортера. Является Open Source-проектом.

2. **Node-exporter** (DaemonSet) — экспортер Prometheus, запускаемый на каждом узле кластера и собирающий метрики с этих узлов. Он собирает данные об операционной системе и аппаратном обеспечении.

Состоит из следующих контейнеров:

- **node-exporter** — основной контейнер. Является Open Source-проектом;

- **kubelet-eviction-thresholds-exporter** — сайдкар-контейнер, который снимает показатели доступности файловых дескрипторов, сокетов, свободного места и inode на каждом узле, сравнивает их в процентном соотношении с Eviction thresholds (порогами вытеснения), установленными в конфигурации kubelet, и экспортирует полученные значения в виде метрик. Является разработкой компании «Флант»;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам экспортера. Подробно описан выше.

3. **Oom-kills-exporter** (DaemonSet) — экспортер Prometheus, запускаемый на каждом узле кластера, следящий за событиями OOM-kill (Out-Of-Memory kill), происходящими с контейнерами, запущенными на этих узлах, и экспортирующий счетчик таких событий в виде метрики. Oom-kills-exporter также следит за подами для синхронизации лейблов, содержащих идентификаторы контейнеров, которые добавляются к метрикам.

Состоит из следующих контейнеров:

- **oom-kills-exporter** — основной контейнер. Является разработкой компании «Флант»;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам экспортера. Подробно описан выше.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- следит за объектами API Kubernetes;
- выполняет авторизацию запросов на метрики.

С модулем взаимодействуют следующие внешние компоненты:

1. Prometheus-main — собирает метрики с экспортеров модуля.

Дополнительные ресурсы

- Документация модуля `monitoring-kubernetes`

Модуль upmeter

Модуль `upmeter` проверяет доступность Deckhouse Kubernetes Platform (DKP) и состояние компонентов кластера в реальном времени. Результаты проверок отображаются на дашбордах.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `upmeter` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:

Номерами на схеме отмечен порядок прохождения запроса пользователя к компонентам `status` и `webui`. На шагах 1, 2 и 3 реализуется обработка запроса Ingress NGINX Controller с обязательной аутентификацией пользователя в единой системе аутентификации платформы, реализуемой модулем `user-authn`. Подробнее с описанием архитектуры модуля `user-authn` можно ознакомиться в соответствующем разделе документации.

Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Upmeter** (StatefulSet) — контроллер, выполняющий следующие операции:

- отслеживает кастомный ресурс Downtime и рассчитывает доступность компонентов DKP на основе интервалов простоя в этом ресурсе;
- хранит метрики доступности компонентов DKP в локальной базе SQLite;
- принимает и обрабатывает данные проверок компонентов DKP;
- обрабатывает API-запросы на получение информации о доступности платформы;
- отслеживает кастомный ресурс UpmeterRemoteWrite и отправляет результаты проб на эндпоинт, указанный в этом ресурсе, по протоколу Prometheus Remote Write.

Состоит из следующих контейнеров:

- **migrator** — init-контейнер, применяющий миграцию базы данных SQLite компонента;
- **upmeter** — основной контейнер;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к API upmeter.

2. **Upmeter-agent** (DaemonSet) — компонент, работающий на master-узлах кластера и выполняющий на регулярной основе следующие группы проверок:

- control-plane — проверка доступности Kubernetes API, а также проверки работы контроллеров control plane кластера;
- deckhouse — проверка состояния кластера DKP, а также проверка работоспособности deckhouse-контроллера модуля deckhouse;
- extensions — проверки, что у всех расширений есть хотя бы один под в состоянии Ready;
- load-balancing — проверки доступности сервисов, обеспечивающих сетевую балансировку;
- monitoring-and-autoscaling — проверки того, что подсистема Observability исправно работает и собирает метрики с системных компонентов;
- nginx — проверки того, что у всех Ingress-контроллеров есть хотя бы один под в состоянии Ready;
- nodegroups — проверка количества `desired` узлов в каждой NodeGroup;
- synthetic — проверка сетевого взаимодействия между узлами кластера через HTTP-запросы к компоненту smoke-mini-[a-e].

В группу проверок control-plane входят следующие пробы:

- apiserver — upmeter-agent проверяет доступность Kubernetes API;
- basic-functionality — upmeter-agent проверяет базовую работу Kubernetes API через жизненный цикл ConfigMap;
- namespace — upmeter-agent создаёт неймспейс `upmeter-probe-namespace` и после проверки удаляет его;
- scheduler — upmeter-agent создаёт под с именем `upmeter-probe-scheduler`, проверяет, что он назначен на какой-либо узел кластера, и удаляет его;
- controller-manager — upmeter-agent создаёт StatefulSet с именем `upmeter-probe-controller-manager` и с заведомо несуществующим контейнером в спецификации создаваемого пода, проверяет, что нужный под создан и находится в ожидаемом состоянии, после чего удаляет StatefulSet;
- cert-manager — upmeter-agent создаёт ресурс Certificate (самоподписанный) с именем `upmeter-probe-cert-manager`, проверяет, что cert-manager сформировал соответствующий ресурс Secret, после чего выполняет удаление созданных ресурсов Certificate и Secret.

Для проверки работоспособности deckhouse-контроллера выполняется следующая последовательность действий:

- upmeter-agent создаёт или обновляет кастомный ресурс UpmeterHookProbe;
- deckhouse-контроллер следит за этим ресурсом и выполняет хук для обновления ресурса;
- upmeter-agent также следит за ресурсом UpmeterHookProbe и при изменении проверяет их корректность.

Проверки или группы проверок могут быть отключены в параметре `.spec.settings.disabledProbes` модуля.

Собранные результаты проб upmeter-agent отправляет через HTTP-запрос `POST /downtime` в upmeter.

Состоит из следующих контейнеров:

- **chown-volume-data** — init-контейнер, устанавливающий необходимые права доступа на каталог `/var/lib/upmeter/agent` на узле кластера;
- **migrator** — init-контейнер, применяющий миграцию базы данных SQLite компонента;
- **agent** — основной контейнер.

3. **Smoke-mini-[a-e]** (StatefulSet) — компонент для синтетических проверок связности. Он состоит из одного контейнера **smoke-mini**. При получении запроса от upmeter-agent экземпляры smoke-mini-[a-e] выполняют запросы друг к другу и к DNS-сервису кластера, после чего возвращают результаты проверок.

При установке модуля `upmeter`, deckhouse-контроллер модуля `deckhouse` регистрирует хук, который распределяет экземпляры StatefulSet по разным узлам кластера (при такой возможности) и после этого каждую минуту перераспределяет по одной реплике StatefulSet на другой узел.

4. **Status** (Deployment) — компонент, состоящий из одного контейнера **status** и реализующий веб-страницу с текущим статусом доступности всех компонентов DKP.
5. **Webui** (Deployment) — компонент, состоящий из одного контейнера **webui** и реализующий дашборд для просмотра истории доступности каждого компонента DKP.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver:**
 - управление кастомными ресурсами UpmeterRemoteWrite и UpmeterHookProbe;
 - обработка кастомных ресурсов Downtime;
 - авторизация запросов к upmeter;
 - создание, проверка и удаление стандартных ресурсов Pod, StatefulSet, Namespace, Secret и кастомных ресурсов Certificate.
2. **Внешние системы хранения метрик** — отправка результатов проверок по протоколу Prometheus Remote Write.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — использование правила мониторинга и метрик, связанных с модулем upmeter.
2. **Controller nginx** — пересылка внешних запросов пользователя к веб-интерфейсу модуля.

Дополнительные ресурсы

- Документация модуля `upmeter`

Модули логирования

Модуль log-shipper

Модуль `log-shipper` упрощает настройку сбора логов в Kubernetes-кластере. Он позволяет организовать сбор логов как с приложений, запущенных в кластере, так и с самих узлов, а затем отправлять их в любую систему хранения — внутреннюю или внешнюю (например, Loki, Elasticsearch и другие).

i

Deckhouse Kubernetes Platform (DKP) обеспечивает интеграцию с системами хранения логов. Сами системы хранения пользователь разворачивает и настраивает самостоятельно.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

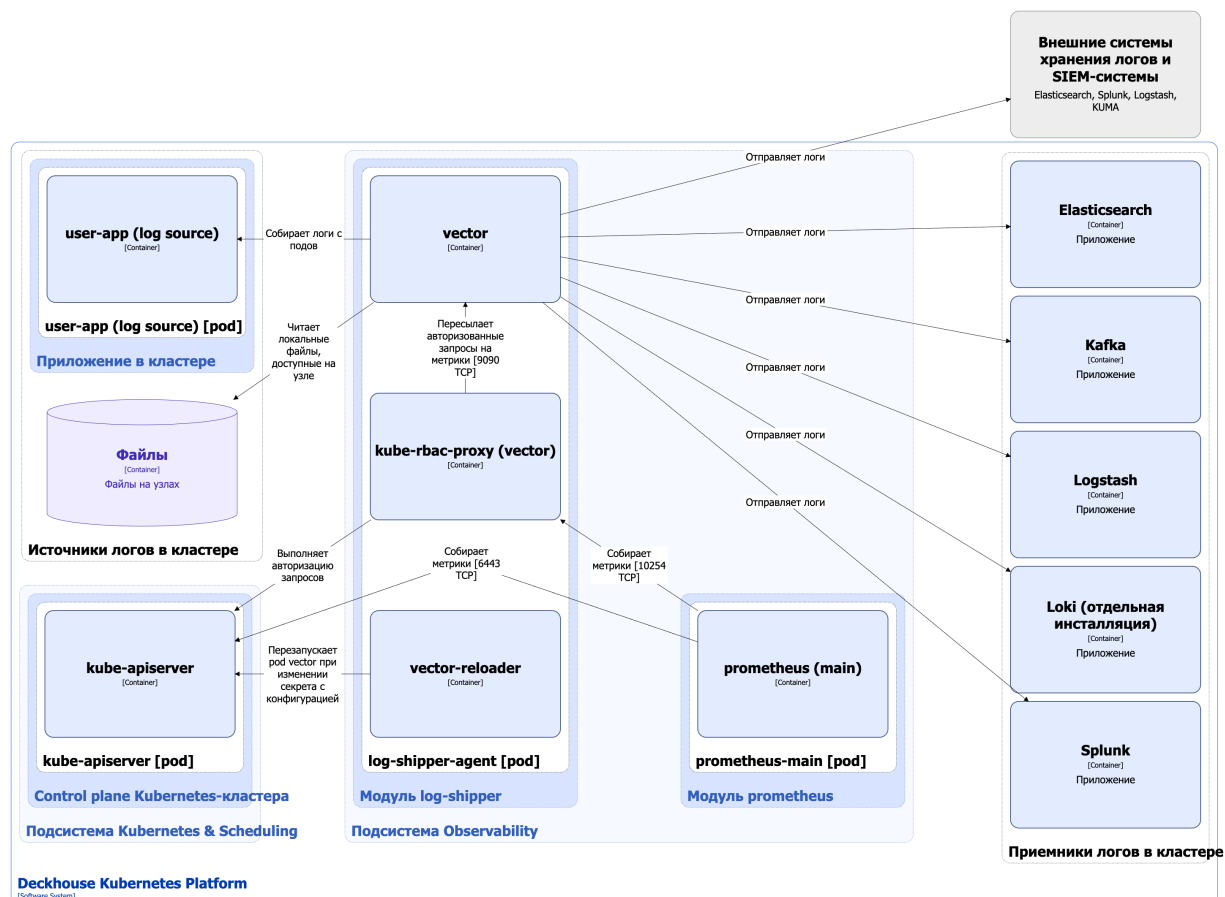
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `log-shipper` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из одного компонента:

- **log-shipper-agent** (DaemonSet) — на каждом узле кластера запускается отдельный экземпляр `log-shipper-agent`, который в свою очередь состоит из следующих контейнеров:
 - **vector** — агент логирования на базе Datadog Vector. Настраивается с помощью кастомных ресурсов `ClusterLogDestination`, `ClusterLoggingConfig` и `PodLoggingConfig`.
 - **vector-reloader** — сайдкар-контейнер Reloader, который отслеживает изменения секрета с конфигурацией. При наличии изменений он проверяет обновленную конфигурацию и перезапускает `vector` для применения обновлений.
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам агента.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Источники логов в кластере:

- приложения, запущенные в кластере — собирает логи с подов;
- файлы — читает локальные файлы на узлах кластера.

2. Приемники логов:

- внутренние системы хранения логов;
- внешние системы хранения логов и SIEM-системы.

В качестве внутренних и внешних приемников логов могут использоваться Elasticsearch, Kafka, Logstash, Loki, Splunk.

3. Kube-apiserver:

- авторизация запросов на получение метрик;
- отслеживание изменений секрета с конфигурацией vector.

С модулем взаимодействуют следующие внешние компоненты:

1. **Prometheus-main** — сбор метрик log-shipper-agent.

Модуль loki

В Kubernetes системные логи на узлах хранятся недолго и могут быть утеряны при перезапуске или обновлении узлов. Модуль `loki` разворачивает в кластере собственное хранилище оперативных логов на базе Grafana Loki.

Возможности модуля:

- системные логи автоматически попадают в Loki без дополнительной настройки;
- доступ к логам реализован через Grafana и веб-интерфейс Deckhouse (модуль `console`).

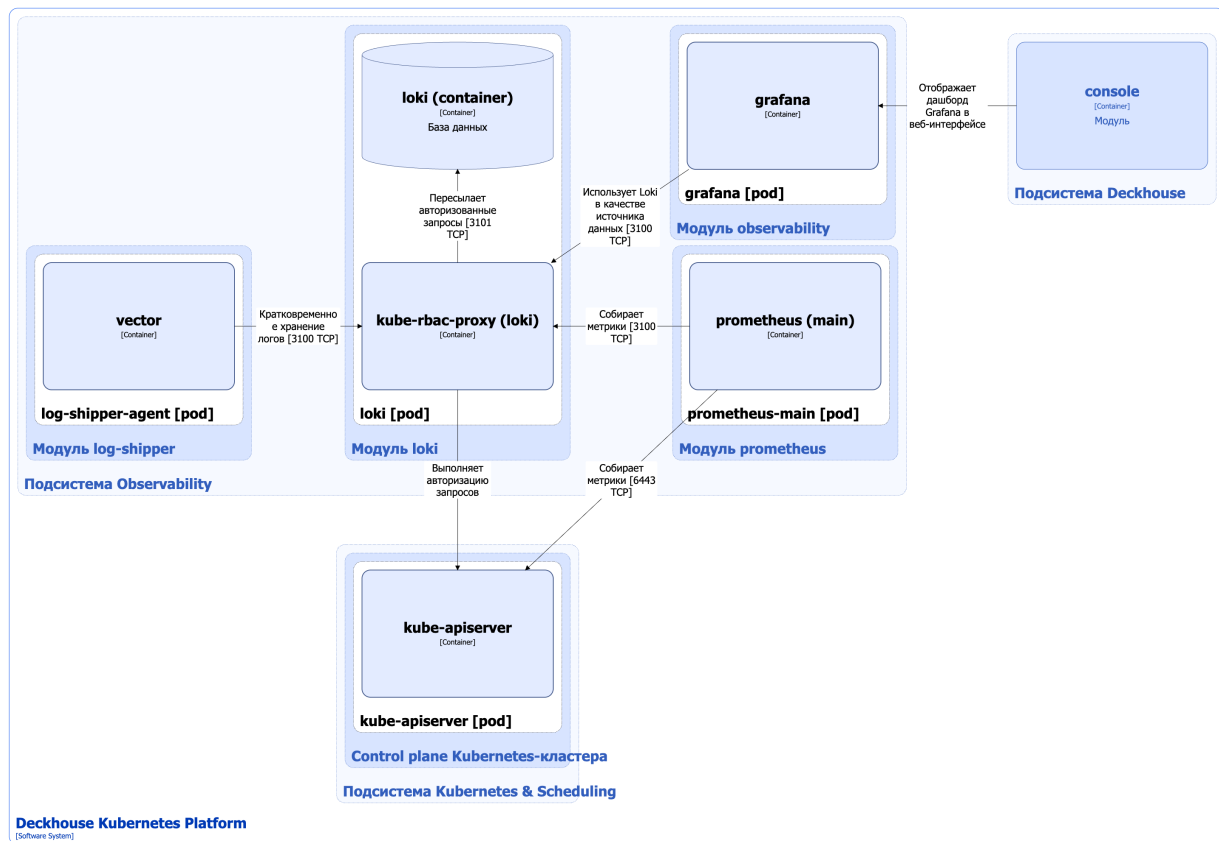


Кратковременное хранилище на базе Grafana Loki не поддерживает работу в режиме высокой доступности (HA). Для долговременного хранения важных логов используйте внешние системы, поддерживаемые модулем `log-shipper`.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

Архитектура модуля

Архитектура модуля `loki` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты модуля

Модуль состоит из одного компонента:

- **loki** — *StatefulSet* из одной реплики **loki-0**, которая в свою очередь состоит из следующих контейнеров:
 - **loki** — контейнер с Grafana Loki;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к loki и его метрикам.

Взаимодействия модуля

С модулем взаимодействуют следующие внешние компоненты:

- **vector** — отправка логов с системных компонентов в loki;
- **console** (встроенная в веб-интерфейс Grafana) — использует loki в качестве источника данных для визуализации и анализа логов;
- **prometheus-main** — сбор метрик loki.

Дополнительные ресурсы

- Документация модуля `log-shipper`
- Документация модуля `loki`

Подсистема Managed Services

В данном подразделе описывается архитектура подсистемы Managed Services в Deckhouse Kubernetes Platform (DKP). Подсистема Managed Services автоматизирует развёртывание, масштабирование, резервное копирование и обновление управляемых сервисов в DKP.

В подсистему Managed Services входят следующие модули:

- `managed-cassandra` — управляет кластерами Cassandra;
- `managed-clickhouse` — управляет экземплярами ClickHouse;
- `managed-hive-metastore` — управляет кластерами Hive Metastore;
- `managed-kafka` — управляет инстансами Kafka;
- `managed-memcached` — управляет инстансами Memcached;
- `managed-opensearch` — управляет инстансами OpenSearch;
- `managed-postgres` — управляет кластерами PostgreSQL;

- `managed-starrocks` — управляет кластерами StarRocks;
- `managed-trino` — управляет кластерами Trino;
- `managed-valkey` — управляет кластерами Valkey.

Дополнительные ресурсы

- Документация модуля `managed-cassandra`
- Документация модуля `managed-clickhouse`
- Документация модуля `managed-hive-metastore`
- Документация модуля `managed-kafka`
- Документация модуля `managed-memcached`
- Документация модуля `managed-postgres`

Модуль managed-memcached

Модуль `managed-memcached` упрощает развертывание и управление Memcached инстансами в Deckhouse Kubernetes Platform (DKP). Он предоставляет:

- **Автоматическое развертывание** — развертывание Memcached-инстансов при помощи простой YAML-конфигурации;
- **Высокая доступность** — поддержка как одиночных (Standalone), так и групповых (Group) развертываний;
- **Управление конфигурацией** — гибкая настройка с валидацией и ограничениями через MemcachedClass;
- **Управление ресурсами** — автоматическое распределение ресурсов и масштабирование;
- **Мониторинг** — встроенное отслеживание состояния инстанса и мониторинг самого сервера Memcached;
- **Безопасность** — distroless-образы и применение паттернов безопасной разработки;
- **Валидация** — CEL-правила для проверки конфигурации.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

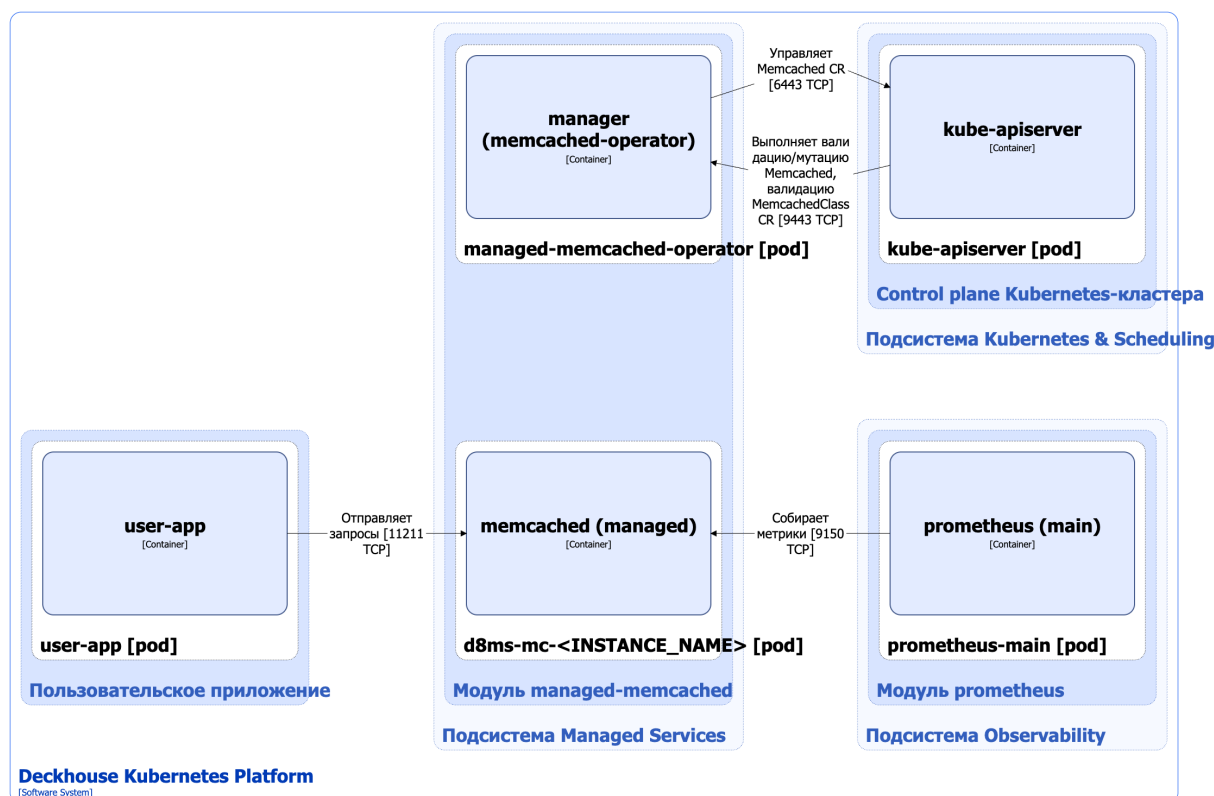
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `managed-memcached` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображена на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Managed-memcached-operator** — оператор Kubernetes, состоящий из одного контейнера **manager** и выполняющий следующие операции:
 - согласование состояния кастомных ресурсов Memcached во всех пользовательских неймспейсах. Ресурс Memcached определяет настройки инстанса Memcached, включая топологию размещения и тип развёртывания;
 - валидация кастомных ресурсов Memcached и MemcachedClass, мутация кастомных ресурсов Memcached с помощью механизма Validating/Mutating Admission Controllers.
2. **d8ms-mc-<INSTANCE_NAME>** (StatefulSet) — один или несколько инстансов Memcached в зависимости от типа развёртывания. Создаются компонентом managed-memcached-operator.

Состоит из одного контейнера:

- **memcached** — является Open Source-проектом.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. **Kube-apiserver** — управляет кастомными ресурсами Memcached.

С модулем взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — отправляет запросы на валидацию кастомных ресурсов Memcached и MemcachedClass, мутацию кастомных ресурсов Memcached.
2. **Prometheus-main** — собирает метрики инстансов Memcached.
3. **Пользовательские приложения** — отправляют запросы к инстансам Memcached.

Дополнительные ресурсы

- Документация модуля `managed-memcached`

Модуль managed-postgres

Модуль `managed-postgres` управляет кластерами PostgreSQL в Deckhouse Kubernetes Platform (DKP). Он позволяет пользователям конфигурировать и масштабировать PostgreSQL-кластеры в соответствии с их потребностями, обеспечивая оптимальную производительность и безопасность. Основные возможности модуля:

- **Автоматическое развертывание** — разворачивает инстанс PostgreSQL при помощи простой YAML-конфигурации;
- **Высокая доступность** — поддерживает установку отказоустойчивого кластера PostgreSQL или одиночного инстанса на выбор;
- **Управление конфигурацией** — отдельный кастомный ресурс `PostgresClass` для шаблонизации подхода к созданию кластеров с возможностью гибко валидировать пользовательские конфигурационные файлы;
- **Управление пользователями и базами данных** — декларативная модель создания пользователей и логических баз данных;
- **Отображение статуса** — информативный набор состояний для отслеживания развернутого PostgreSQL.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в соответствующем разделе документации.

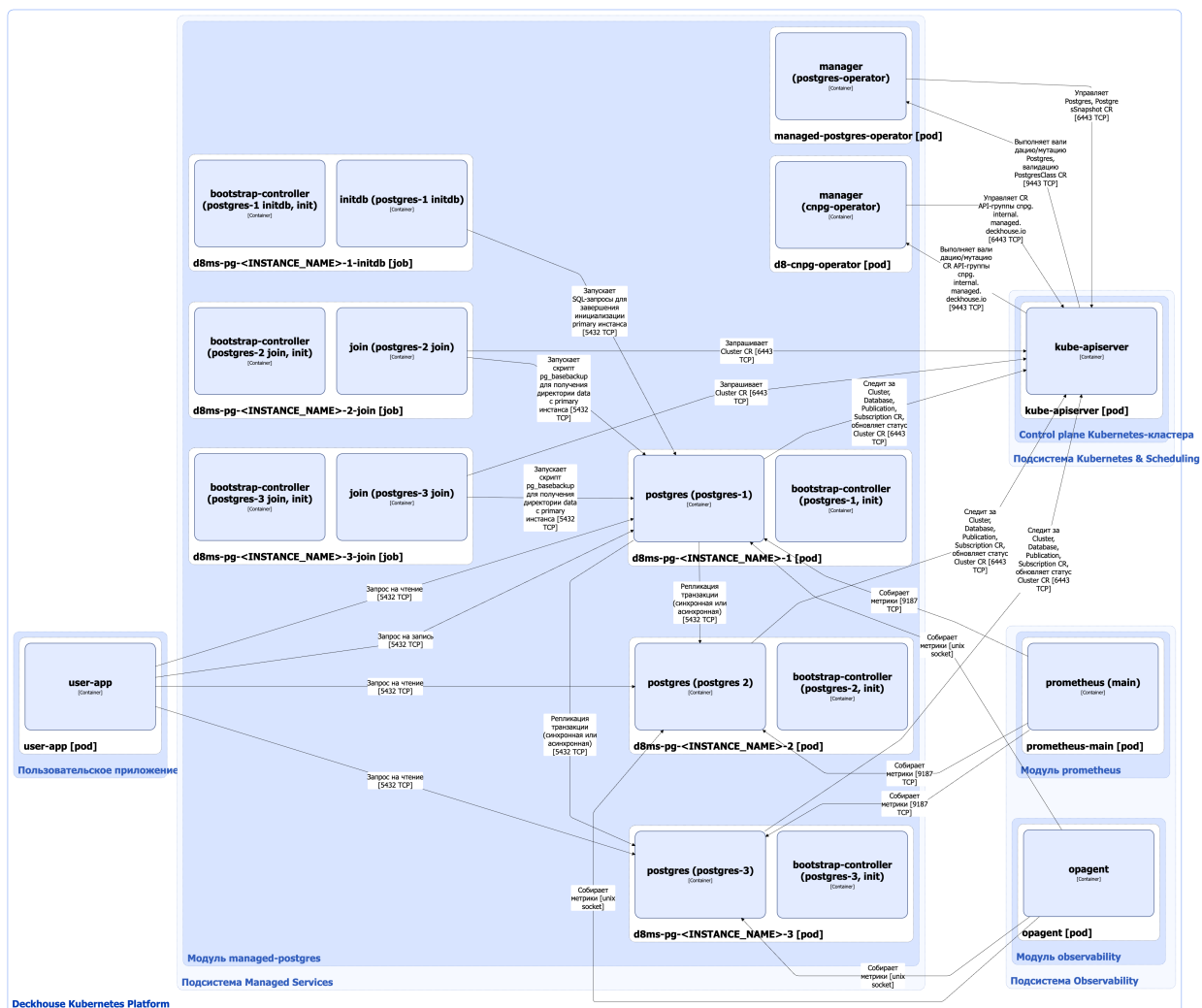
Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.

Архитектура модуля `managed-postgres` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображена на следующей диаграмме:



Компоненты модуля

Модуль состоит из следующих компонентов:

1. **Managed-postgres-operator** — оператор Kubernetes, состоящий из одного контейнера **manager** и выполняющий следующие операции:
 - согласование состояния кастомных ресурсов Postgres во всех пользовательских неймспейсах. Ресурс Postgres определяет настройки кластера PostgreSQL, включая топологию размещения и режим репликации, конфигурацию инстансов PostgreSQL и прочие параметры, такие как списки логических баз данных и внутренних пользователей;
 - согласование состояния кастомных ресурсов PostgresSnapshot. Ресурс PostgresSnapshot предназначен для резервного копирования и восстановления инстансов PostgreSQL;
 - валидация кастомных ресурсов Postgres и PostgresClass, мутация кастомных ресурсов Postgres с помощью механизма Validating/Mutating Admission Controllers.

В качестве бэкенда Managed-postgres-operator использует кастомные ресурсы из API-группы `cnpg.internal.managed.deckhouse.io`, управляемые компонентом d8-cnpg-operator.

2. **D8-cnpg-operator** — это форк CloudNativePG, оператора Kubernetes, который автоматизирует управление кластерами PostgreSQL. D8-cnpg-operator состоит из одного контейнера **manager** и выполняет следующие операции:
 - управление кастомными ресурсами из API-группы `cnpg.internal.managed.deckhouse.io`:
 - Cluster — определяет кластер PostgreSQL;
 - Pooler — определяет настройки пула соединений;
 - FailoverQuorum — служит для отображения состояния кворума реплик кластера PostgreSQL в отказоустойчивой конфигурации;
 - Database — определяет логическую базу данных;
 - Backup — определяет резервную копию инстанса PostgreSQL;
 - ScheduledBackup — определяет настройки резервного копирования PostgreSQL по расписанию;
 - Subscription — определяет принимающую сторону логической репликации;
 - Publication — определяет источник для логической репликации;
 - ImageCatalog — определяет образы PostgreSQL для пользовательских пространств имён;
 - ClusterImageCatalog — определяет образы PostgreSQL для всего кластера Kubernetes.
 - валидация и мутация кастомных ресурсов из API-группы `cnpg.internal.managed.deckhouse.io` помощью механизма Validating/Mutating Admission Controllers.

i

Ниже описаны компоненты для режима репликации ConsistencyAndAvailability. Два других режима репликации по составу компонентов являются частным случаем режима ConsistencyAndAvailability.

3. **d8ms-pg-<INSTANCE_NAME>-1-initdb** (Job) — задание, создаваемое компонентом d8-cnpg-operator, которое запускает SQL-запросы для завершения инициализации primary инстанса PostgreSQL.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
 - **initdb** — основной контейнер, в котором запускается исполняемый файл `manager`, выполняющий описанные выше SQL-запросы.
4. **d8ms-pg-<INSTANCE_NAME>-2-join** (Job) — задание, создаваемое компонентом d8-cnpg-operator, которое запускает скрипт `pg_basebackup` для получения директории `data` с primary инстанса для первой реплики. Задание использует кастомный ресурс Cluster для настройки реплики.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
- **join** — основной контейнер, в котором запускается исполняемый файл `manager`,

выполняющий описанный выше скрипт.

5. **d8ms-pg-<INSTANCE_NAME>-3-join** (Job) — задание, создаваемое компонентом d8-cnpg-operator, которое запускает скрипт `pg_basebackup` для получения директории `data` с primary инстанса для второй реплики. Задание использует кастомный ресурс Cluster для настройки реплики.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
- **join** — основной контейнер, в котором запускается исполняемый файл `manager`, выполняющий описанный выше скрипт.

6. **d8ms-pg-<INSTANCE_NAME>-1** — primary инстанс PostgreSQL. Создается компонентом d8-cnpg-operator.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
- **postgres** — основной контейнер, в котором запускается исполняемый файл `manager`, который выполняет следующие операции:
 - старт процессов PostgreSQL;
 - управление жизненным циклом экземпляра PostgreSQL, включая мониторинг состояния сервера, обработку остановки и перезапуска сервера;
 - участие в процедурах `switchover/failover`. `Switchover` — это плановый, контролируемый процесс, при котором активный первичный инстанс (primary) намеренно выводится из эксплуатации, а назначенный резервный инстанс (реплика) повышается до роли первичного. Его главная цель — обеспечить нулевую потерю данных: перед тем как передать роль реплике, мы ждём, пока все текущие транзакции реплицируются. `Failover` — это аварийная ситуация: первичный инстанс вышел из строя, стал недоступен или его нельзя безопасно использовать для записи. В этом случае назначенный резервный инстанс (реплика) повышается до роли первичного, при этом возможна потеря данных;
 - взаимодействие с оператором и публикация состояния экземпляра;
 - отслеживание изменений кастомных ресурсов Cluster, Database, Publication, Subscription.

7. **d8ms-pg-<INSTANCE_NAME>-2** — первая реплика PostgreSQL. Создается компонентом d8-cnpg-operator.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
- **postgres** — основной контейнер, в котором запускается исполняемый файл `manager`, выполняющий операции, описанные для primary инстанса PostgreSQL.

8. **d8ms-pg-<INSTANCE_NAME>-3** — вторая реплика PostgreSQL. Создается компонентом d8-cnpg-operator.

Состоит из следующих контейнеров:

- **bootstrap-controller** — init-контейнер, выполняющий установку исполняемого файла `manager` компонента D8-cnpg-operator;
- **postgres** — основной контейнер, в котором запускается исполняемый файл `manager`, выполняющий операции, описанные для primary инстанса PostgreSQL.

Взаимодействия модуля

Модуль взаимодействует со следующими компонентами:

1. Kube-apiserver:

- управляет кастомными ресурсами Postgres и PostgresSnapshot;
- управляет кастомными ресурсами API-группы `cnpg.internal.managed.deckhouse.io`.

С модулем взаимодействуют следующие внешние компоненты:

1. Kube-apiserver:

- отправляет запросы на валидацию кастомных ресурсов Postgres и PostgresClass, мутацию кастомных ресурсов Postgres;

- отправляет запросы на валидацию и мутацию кастомных ресурсов из `cnpg.internal.managed.deckhouse.io` API-группы.
- 2. **Prometheus-main** — собирает метрики экземпляров кластера PostgreSQL (которые получает исполняемый файл `manager` компонента `d8-cnpg-operator`, запущенный в контейнере `postgres` пода экземпляра PostgreSQL).
- 3. **opAgent** — собирает метрики экземпляров кластера PostgreSQL (подключаясь к экземплярам PostgreSQL напрямую) и отправляет их в `prometheus-main`.
- 4. **Пользовательские приложения** — отправляют запросы к экземплярам кластера PostgreSQL. Запросы на запись отправляются в `primary` экземпляр через сервис Kubernetes `d8ms-pg-<INSTANCE_NAME>-rw`. `Primary` экземпляр реплицирует транзакции в реплики. Запросы на чтение балансируются на экземпляры PostgreSQL через сервисы Kubernetes `d8ms-pg-<INSTANCE_NAME>-r` или `d8ms-pg-<INSTANCE_NAME>-ro`.

Дополнительные ресурсы

- Документация модуля `managed-postgres`

Модуль managed-valkey

Модуль `managed-valkey` управляет экземплярами Valkey (Redis-совместимое хранилище данных в оперативной памяти) в Deckhouse Kubernetes Platform (DKP). Он предоставляет:

- **Автоматическое развёртывание** — создаёт экземпляр Valkey при помощи простой YAML-конфигурации;
- **Standalone** — поддерживает установку одиночного экземпляра;
- **Persistent Storage** — позволяет сконфигурировать разные варианты хранения данных: `AOF`, `RDB`, `AOF+RDB`;
- **Управление конфигурацией** — отдельный ресурс `ValkeyClass` для шаблонизации создания сервиса и гибкой валидации пользовательских параметров;
- **Статус** — отображает текущее состояние развёрнутого экземпляра Valkey.

Подробнее с настройками модуля и примерами его использования можно ознакомиться в документации модуля.

Архитектура модуля

i

Для упрощения схемы приняты следующие допущения:

- На схеме показано, что контейнеры разных подов взаимодействуют друг с другом напрямую. Фактически они взаимодействуют через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса указано над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме все поды изображены в одной реплике.

Архитектура модуля `managed-valkey` на уровне 2 модели C4 и его взаимодействие с другими компонентами DKP изображены на следующей диаграмме:

Компоненты модуля

Модуль состоит из следующих компонентов:

1. `managed-valkey-operator` (Deployment) — оператор Kubernetes, состоящий из одного контейнера `manager` и выполняющий следующие операции:
 - согласование состояния кастомных ресурсов Valkey во всех пользовательских неймспейсах. Ресурс Valkey определяет настройки экземпляра Valkey;
 - создание и управление ресурсами `StatefulSet`, `Secret`, `ConfigMap` и `PersistentVolumeClaim`, относящимися к экземпляру Valkey.
2. `managed-valkey-webhook` (Deployment) — компонент, состоящий из одного контейнера `manager`.

Компонент выполняет валидацию и мутацию кастомных ресурсов Valkey, а также мутацию

кастомных ресурсов ValkeyClass с помощью механизмов Validating Admission Controller и Mutating Admission Controller.

3. `d8ms-valkey-<INSTANCE_NAME>` (StatefulSet) — компонент выполняет запуск и подготовку инстанса Valkey. Создаётся компонентом `managed-valkey-operator`.

Состоит из двух контейнеров:

- `valkey` — основной контейнер, являющийся Open Source-проектом;
- `agent` — сайдкар-контейнер, выполняющий настройку основного контейнера в соответствии с параметрами в ресурсе Valkey.

Взаимодействия модуля

Модуль взаимодействует с компонентом `kube-apiserver`, через который:

- управляет кастомными ресурсами Valkey, ValkeyClass и Certificate;
- управляет ресурсами StatefulSet, Secret, ConfigMap и PersistentVolumeClaim.

С модулем взаимодействуют следующие внешние компоненты:

1. `kube-apiserver` — обрабатывает запросы на валидацию и мутацию кастомных ресурсов Valkey.
2. `prometheus-main` — собирает метрики компонентов `managed-valkey-operator` и `managed-valkey-webhook`.
3. Пользовательские приложения — отправляют запросы к инстансу Valkey.

Дополнительные ресурсы

- Документация модуля `managed-valkey`

Подсистема Virtualization

В данном подразделе описана архитектура подсистемы Virtualization (подсистемы виртуализации) Deckhouse Kubernetes Platform (DKP).

Подсистема Virtualization представлена модулем `virtualization`, который позволяет декларативно создавать, запускать и управлять виртуальными машинами (VM) и их ресурсами.

Модуль `virtualization` включает в себя следующие компоненты:

- Virtualization API (API) — контроллер, реализующий API пользователя для создания и управления ресурсами виртуальных машин;
- ядро модуля — основано на проекте KubeVirt и использует QEMU/KVM + libvirt для запуска VM. KubeVirt — это Open Source-проект, который позволяет запускать, развёртывать и управлять VM с использованием Kubernetes в качестве платформы оркестрации. Он обеспечивает совместную работу традиционных VM и контейнерных рабочих нагрузок в одном кластере Kubernetes, предоставляя единую плоскость управления;
- Deckhouse Virtualization Container Registry (DVCR) — хранилище образов контейнеров для хранения и кеширования образов VM;
- Containerized Data Importer (CDI) — дополнение для управления постоянным хранилищем в Kubernetes. Его основная цель — предоставить декларативный способ создания дисков VM на основе ресурсов PersistentVolumeClaim (PVC). CDI предоставляет возможность импортировать образы и диски VM в PVC-тома для использования в VM, управляемых KubeVirt;
- вспомогательные компоненты — компоненты, реализующие следующие вспомогательные функции:
 - аудит событий безопасности;
 - проброс USB-устройств в VM;
 - обновление сетевых маршрутов;
 - удаление ресурсов перед деактивацией модуля `virtualization`.

Подробнее с описанием модуля можно ознакомиться в разделе документации модуля.

Дополнительные ресурсы

- Документация модуля `virtualization`

Ядро модуля

Ядро модуля `virtualization` непосредственно отвечает за работу с виртуальными машинами (ВМ). Ядро основано на проекте KubeVirt. KubeVirt — это Open Source-проект, который позволяет запускать, развёртывать и управлять ВМ с использованием Kubernetes в качестве платформы оркестрации. Он обеспечивает совместную работу традиционных ВМ и контейнерных рабочих нагрузок в одном кластере Kubernetes, предоставляя единую плоскость управления. В модуле `virtualization` используется форк KubeVirt от компании «Флант».

Для управления ВМ ядро модуля использует кастомные ресурсы следующих API-групп:

1. `internal.virtualization.deckhouse.io` — основная группа, аналог API-группы `kubvirt.io` оригинального KubeVirt. Включает следующие кастомные ресурсы:
 - `InternalVirtualizationVirtualMachine` — описание конфигурации и статуса ВМ;
 - `InternalVirtualizationVirtualMachineInstance` — описание работающей ВМ. После выключения ВМ ресурс `InternalVirtualizationVirtualMachineInstance` удаляется, но остаётся ресурс `InternalVirtualizationVirtualMachine`, который управляет жизненным циклом `InternalVirtualizationVirtualMachineInstance`.

Ресурсами основной группы управляет компонент `virt-controller`. Ресурс `InternalVirtualizationVirtualMachine` основной API-группы `internal.virtualization.deckhouse.io` KubeVirt используется в качестве бэкенда для ресурса `VirtualMachine` API-группы `virtualization.deckhouse.io`, управляемой `virtualization-controller`.

i Для упрощения для ресурсов `InternalVirtualizationVirtualMachine` и `InternalVirtualizationVirtualMachineInstance` далее будут использоваться сокращённые наименования `VirtualMachine` и `VirtualMachineInstance` соответственно (из API-группы `kubvirt.io` оригинального KubeVirt).

2. `subresources.kubvirt.io` — группа субресурсов. Субресурсы — это дополнительные операции или действия, которые можно выполнять над основными ресурсами (например, `VirtualMachineInstance`) через API Kubernetes. Они предоставляют интерфейсы для управления конкретными аспектами ресурсов, не затрагивая весь объект. Вместо привычного для Kubernetes декларативного ресурса они представляют собой эндпоинт для императивных операций. В модуле `virtualization` используются следующие субресурсы KubeVirt:
 - `virtualmachines/{name}/addvolume`;
 - `virtualmachines/{name}/removevolume`;
 - `virtualmachines/{name}/addresourceclaim`;
 - `virtualmachines/{name}/removeresourceclaim`;
 - `virtualmachineinstances/{name}/console`;
 - `virtualmachineinstances/{name}/vnc`;
 - `virtualmachineinstances/{name}/portforward`;
 - `virtualmachineinstances/{name}/freeze`;
 - `virtualmachineinstances/{name}/unfreeze`.

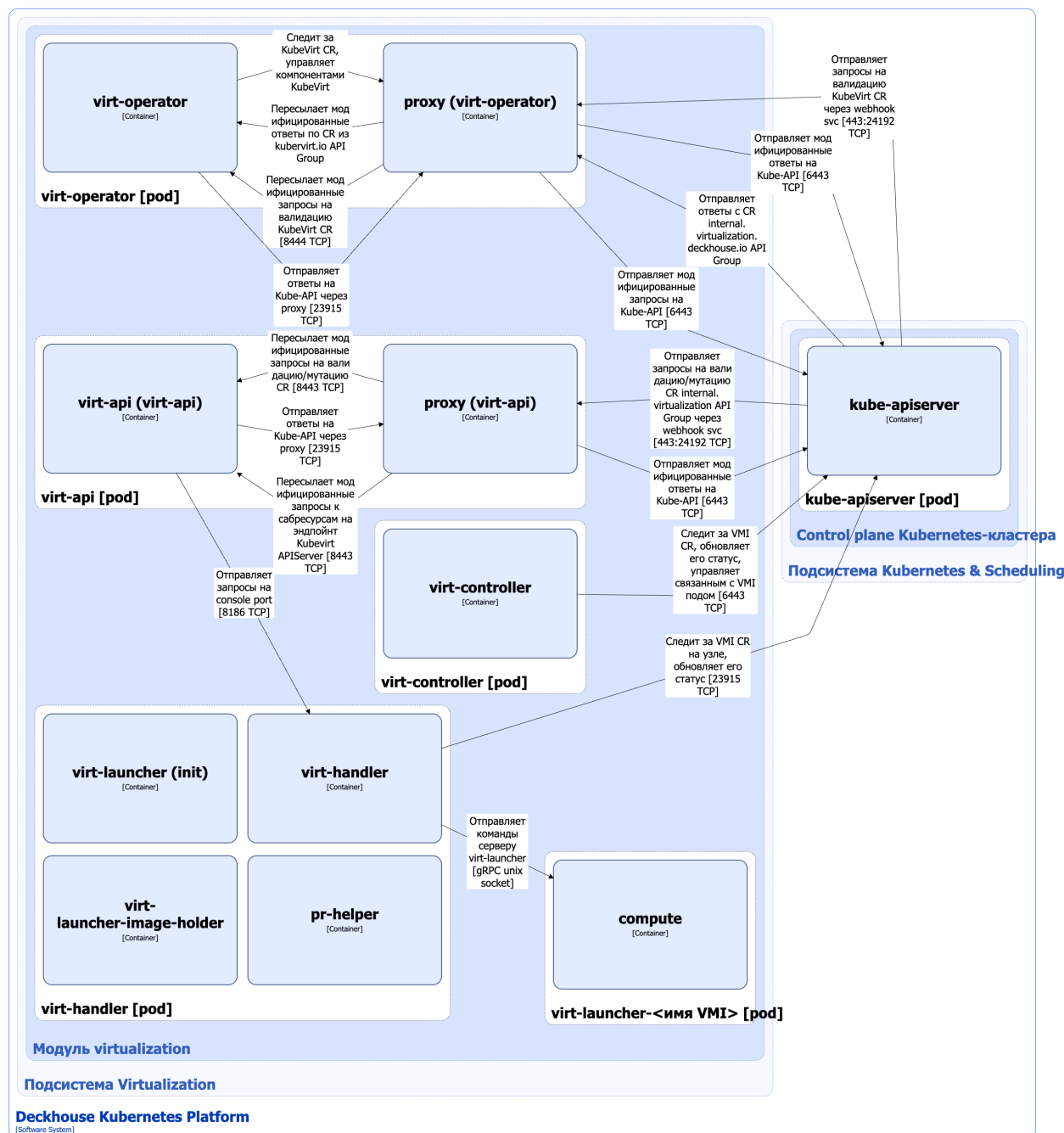
Субресурсами управляет компонент `virt-api`. Перечисленные выше субресурсы KubeVirt используются в качестве бэкенда для аналогичных ресурсов из `subresources.virtualization.deckhouse.io` API-групп, управляемых компонентом `virtualization-api`.

Архитектура ядра модуля

i Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура ядра модуля `virtualization` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты ядра модуля

Ядро модуля состоит из следующих компонентов:

1. **Virt-api** — Kubernetes Extension API Server, обслуживающий запросы к `subresources.kubervirt.io` API-группы. Virt-api выполняет валидацию и мутацию кастомных ресурсов из `internal.virtualization.deckhouse.io` API-группы с помощью механизма Validating/Mutating Admission Controllers. Запросы проходят через сайдкар-контейнер **proxy**, который переименовывает метаданные из API-группы `internal.virtualization.deckhouse.io` в API-группу `kubervirt.io` и проксирует их на эндпоинт virt-api.

Состоит из следующих контейнеров:

- **virt-api** — основной контейнер, реализующий контроллер и вебхук-сервер;
- **proxy** (он же **kube-api-rewriter**) — сайдкар-контейнер, выполняющий модификацию проходящих через него запросов API, а именно переименование метаданных кастомных ресурсов. Это необходимо, поскольку компоненты KubeVirt используют API-группы вида `*.kubervirt.io`, а другие компоненты модуля `virtualization` используют аналогичные ресурсы, но с API-группой вида `*.virtualization.deckhouse.io`. Kube-api-rewriter является шлюзом, проксирующим запросы между контроллерами, управляющими ресурсами из разных API-групп;
- **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes

RBAC для организации защищённого доступа к метрикам контроллера и сайдкар-контейнера `proxu`. Является Open Source-проектом.

2. **Virt-controller** — контроллер, управляющий кастомными ресурсами основной `internal.virtualization.deckhouse.io` API-группы и отвечающий за функциональность виртуализации на уровне кластера (cluster wide). Для каждого ресурса `VirtualMachineInstance` он создаёт отдельный под, в котором запускается ВМ. `Virt-controller` следит за `VirtualMachineInstance` ресурсами, обновляет их статус и управляет связанными с ними подами.

Состоит из следующих контейнеров:

- **virt-controller** — основной контейнер;
- **proxy** (он же **kube-api-rewriter**) — сайдкар-контейнер, выполняющий модификацию проходящих через него запросов API. Подробно описан выше;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступ к метрикам и состоянию контроллера. Подробно описан выше.

3. **Virt-handler** (DaemonSet) — отдельный контроллер, запускающийся на всех узлах кластера. `Virt-handler` выполняет следующие функции:

- расширяет функционал `kubelet`, донстраивая окружение пода для запуска ВМ внутри него. На данный момент `virt-handler` отвечает за создание сетевых интерфейсов, а также используется для проброса `dev/kvm` и других устройств с узла внутрь пода. Для проброса устройств `virt-handler` использует `kubelet device plugins`;
- как и `virt-controller`, `virt-handler` следит за `VirtualMachineInstance` ресурсами, соответствующими запущенным на узле ВМ. При обнаружении изменений `virt-handler` отправляет команду процессу `virt-launcher`, запущенному в контейнере `compute` пода ВМ. `Virt-launcher` изменяет состояние ВМ в соответствии с полученной командой. `Virt-handler` также следит за событиями ВМ, которые возвращает `virt-launcher`, и синхронизирует состояние соответствующего `VirtualMachineInstance` ресурса;
- принимает через `console-port` команды от компонента `virt-api`, соответствующие запросам на субресурсы, и пересылает их на исполнение в `virt-launcher`. Благодаря функционалу субресурсов осуществляется проброс портов до ВМ, а также обычной и VNC-консоли.

`Virt-handler` взаимодействует с `virt-launcher` по gRPC-протоколу через Unix-сокеты.

Состоит из следующих контейнеров:

- **virt-launcher** — `init`-контейнер, запускающий через `virt-launcher` скрипт `node-labeler.sh`. Этот скрипт подготавливает данные по характеристикам процессоров, их функциям и типам машин, которые `virt-handler` будет использовать для установки соответствующих лейблов на ресурсах `Node`. Эти лейблы в свою очередь будут использоваться для планирования ВМ на узлах, которые поддерживают соответствующие параметры;
- **virt-handler** — основной контейнер;
- **virt-launcher-image-holder** — служебный сайдкар-контейнер для предварительного скачивания образа `virt-launcher`. Контейнер стоит на паузе и выполняет только функцию хранения образа;
- **pr-helper** — QEMU persistent reservation helper, служебный сайдкар-контейнер, создаёт сокет-слушатель, который принимает входящие соединения для коммуникации с QEMU. Это необходимо, поскольку операционная система ограничивает отправку команд SCSI с постоянным резервированием непривилегированным программам, что не позволяет совместно использовать блочные SCSI-устройства несколькими ВМ, например в случае кластеризации. QEMU — Open Source-проект для эмуляции аппаратного обеспечения различных платформ, который используется для запуска ВМ в поде.

4. **Virt-operator** — оператор Kubernetes, управляющий жизненным циклом компонентов `KubeVirt` при помощи кастомного ресурса `InternalVirtualizationKubeVirt`. `Virt-operator` устанавливает в кластере `virt-api`, `virt-controller` и `virt-handler`, а также выполняет их настройку.

Состоит из следующих контейнеров:

- **virt-operator** — основной контейнер;
- **proxy** (он же **kube-api-rewriter**) — сайдкар-контейнер, выполняющий модификацию проходящих через него запросов API. Подробно описан выше.

5. **Virt-launcher-[имя VMI]** — под, в котором запускается ВМ (точнее `VirtualMachineInstance`).

Состоит из одного контейнера:

- **compute** — контейнер, в котором запускается virt-launcher. Virt-launcher реализует cmd-server (gRPC-сервер для удалённого выполнения команд).

Virt-launcher в зависимости от поступающей от virt-handler команды формирует XML-спецификацию запускаемой или обновляемой VM и отправляет её в libvirtd. Libvirtd — это демон серверной части системы управления виртуализацией libvirt. Он работает на хост-серверах и выполняет задачи управления для виртуальных гостевых систем.

Libvirtd в свою очередь запускает VM и управляет её жизненным циклом. VM запускается при помощи QEMU и KVM. QEMU — свободно разрабатываемый эмулятор, который поддерживает аппаратную виртуализацию и работает в связке с гипервизором KVM.

Фактически libvirtd запускает процесс QEMU, который и есть VM (точнее VirtualMachineInstance).

Также virt-handler постоянно следит за состоянием запущенной VM, которое возвращает libvirtd через virt-launcher, и обновляет статус VirtualMachineInstance.

Взаимодействия ядра модуля

Ядро модуля взаимодействует со следующими компонентами:

1. Kube-apiserver:

- следит за кастомными ресурсами KubeVirt, управляет компонентами KubeVirt;
- следит за VirtualMachineInstance ресурсами, обновляет их статус и управляет связанными с ними подами;
- выполняет авторизацию запросов на получение метрик.

2. CDI (Containerized-Data-Importer) — KubeVirt на основе спецификации диска и ссылки на образ VM в секции DataVolumeTemplate ресурса VirtualMachine создаёт DataVolume. CDI импортирует в PVC образ диска из указанного в DataVolume источника. Созданный PVC является диском VM, управляемой KubeVirt.

С ядром модуля взаимодействуют следующие внешние компоненты:

1. Kube-apiserver:

- отправляет запросы на валидацию кастомных ресурсов InternalVirtualizationKubeVirt;
- отправляет запросы на валидацию и мутацию кастомных ресурсов из `internal.virtualization.deckhouse.io` API-группы.

2. Prometheus-main — собирает метрики компонентов.

Дополнительные ресурсы

- Документация модуля `virtualization`

Deckhouse Virtualization Container Registry (DVCR)

Компонент Deckhouse Virtualization Container Registry (DVCR) модуля `virtualization` — это специализированное хранилище образов контейнеров для хранения и кеширования образов виртуальных машин (VM). Компонент CDI модуля `virtualization` использует хранящиеся в DVCR образы в качестве источника для ресурсов InternalVirtualizationDataVolume, с помощью которых создаются диски VM, управляемых KubeVirt.

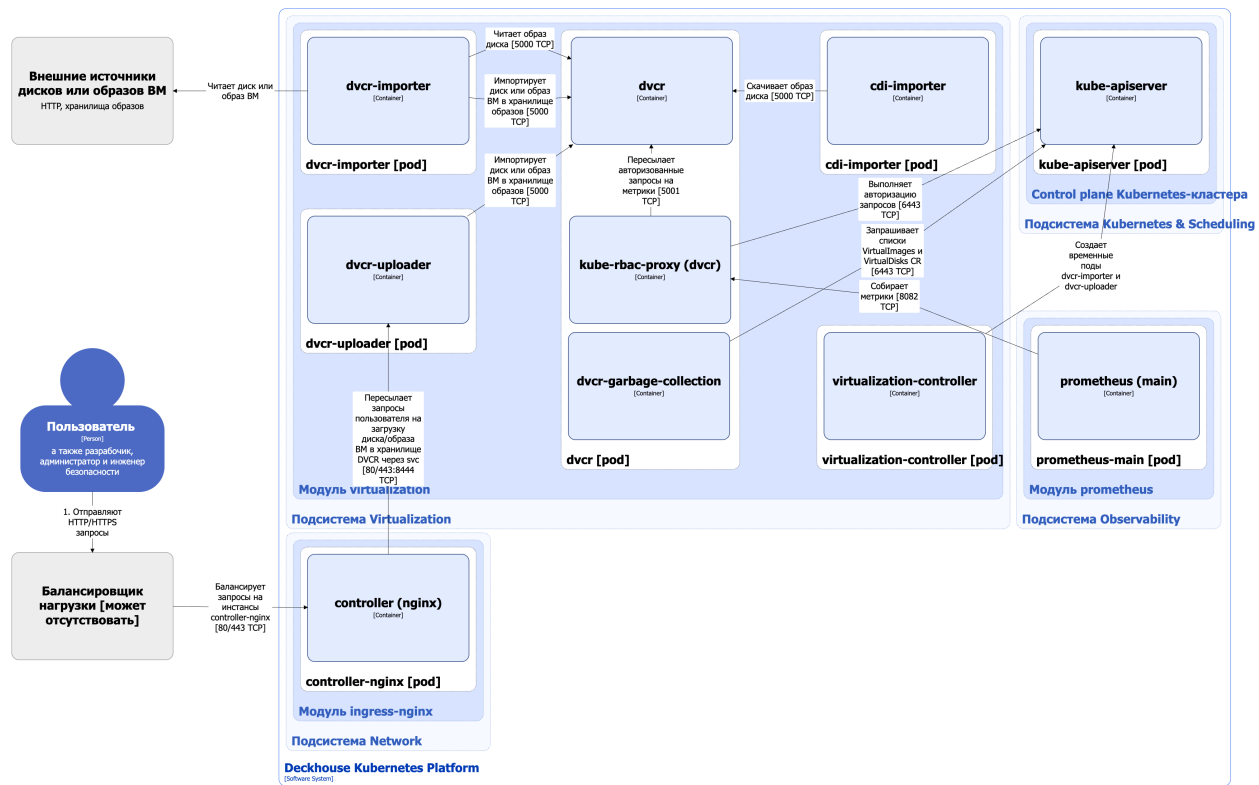
Архитектура DVCR

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура компонента DVCR модуля `virtualization` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты DVCR

DVCR состоит из следующих компонентов:

1. **Dvcr** — хранилище образов на базе Distribution. Distribution — это Open Source-проект, который является основой для хранения и распределения контейнерных образов и другого контента с использованием спецификации OCI Distribution Specification. Dvcr используется для хранения и кеширования образов VM.

Компонент содержит следующие контейнеры:

- **dvcr** — основной контейнер;
 - **dvcr-garbage-collection** — сайдкар-контейнер, выполняющий периодическое удаление образов, у которых нет соответствующих ресурсов в кластере;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера dvcr. Является Open Source-проектом.
2. **Dvcr-importer** — временный под, состоящий из одного контейнера и запускаемый virtualization-controller для реализации различных сценариев импорта образов и дисков VM, таких как:
 - импорт диска или образа VM из внешних источников (HTTP-источник, доступный по URL-ссылке, или хранилище образов) в хранилище DVCR;
 - импорт образа VM из внешних источников (HTTP-источник, доступный по URL-ссылке, или хранилище образов) в PVC-том. Dvcr-importer не импортирует диск напрямую в PVC. Он загружает источник в хранилище DVCR. Далее создаётся ресурс InternalVirtualizationDataVolume, и CDI уже импортирует образ из хранилища DVCR в PVC;
 - импорт образа VM, диска или снимка из ресурсов VirtualImage, ClusterVirtualImage, VirtualDisk или VirtualDiskSnapshot в хранилище DVCR.
 3. **Dvcr-uploader** — временный под, состоящий из одного контейнера и запускаемый virtualization-controller для реализации следующих сценариев загрузки *пользователем* образов и дисков VM:
 - загрузка в хранилище DVCR;
 - загрузка в PVC-том. Dvcr-uploader не загружает диск напрямую в PVC. Он загружает источник в хранилище DVCR. Далее создаётся ресурс InternalVirtualizationDataVolume, и CDI уже импортирует образ из хранилища DVCR в PVC.

Взаимодействия DVCR

DVCR взаимодействует со следующими компонентами:

1. **Kube-apiserver** — выполняет `get/list/watch`-запросы ресурсов `VirtualImages`, `ClusterVirtualImages` и `VirtualDisks` для очистки неиспользуемых образов и координации.
2. **Внешние источники дисков или образов ВМ** — читает диски или образы ВМ при реализации некоторых сценариев импорта в хранилище DVCR.

С DVCR взаимодействуют следующие внешние компоненты:

1. **Virtualization-controller** — запускает поды `dvcr-importer` и `dvcr-uploader` для выполнения сценариев импорта и загрузки дисков и образов ВМ.
2. **Ingress-controller** — пересылает запросы пользователя на загрузку диска или образа ВМ в хранилище DVCR через HTTP-эндпоинт сервиса `dvcr-uploader`.
3. **Cdi-importer** — использует хранящиеся в DVCR образы в качестве источника для ресурсов `InternalVirtualizationDataVolume`.

Дополнительные ресурсы

- Документация модуля `virtualization`

Containerized Data Importer (CDI)

i В модуле `virtualization` используется форк CDI. Оригинальный CDI является подпроектом KubeVirt. KubeVirt — это Open-Source-проект, который позволяет запускать, развёртывать и управлять виртуальными машинами (ВМ) с использованием Kubernetes в качестве платформы оркестрации.

Компонент Containerized Data Importer (CDI) модуля `virtualization` — это дополнение для управления постоянным хранилищем в Kubernetes. Его основная цель — предоставить декларативный способ создания дисков ВМ на основе ресурсов `PersistentVolumeClaim` (PVC). CDI предоставляет возможность импортировать образы и диски ВМ в PVC-тома для использования в ВМ, управляемых KubeVirt. Данные могут поступать из разных источников:

- URL-адрес;
- хранилище образов;
- другой PVC (клонирование);
- снимок;
- загрузка пользователем.

CDI поддерживает импорт данных двух типов:

- **Данные KubeVirt** — импортируемый файл следует рассматривать как диск ВМ KubeVirt. При необходимости CDI автоматически распакует и преобразует файл из поддерживаемых форматов в формат `raw` или `qcow2` (в зависимости от режима тома (`volume mode`)). Также будет изменён размер диска, чтобы использовать всё доступное пространство.
- **Архивные данные** — этот тип означает, что данные хранятся в TAR-архиве. Сжатие архивов не поддерживается. CDI извлекает содержимое архива в том, который затем можно использовать либо с обычным подом, либо с ВМ, используя функцию файловой системы KubeVirt.

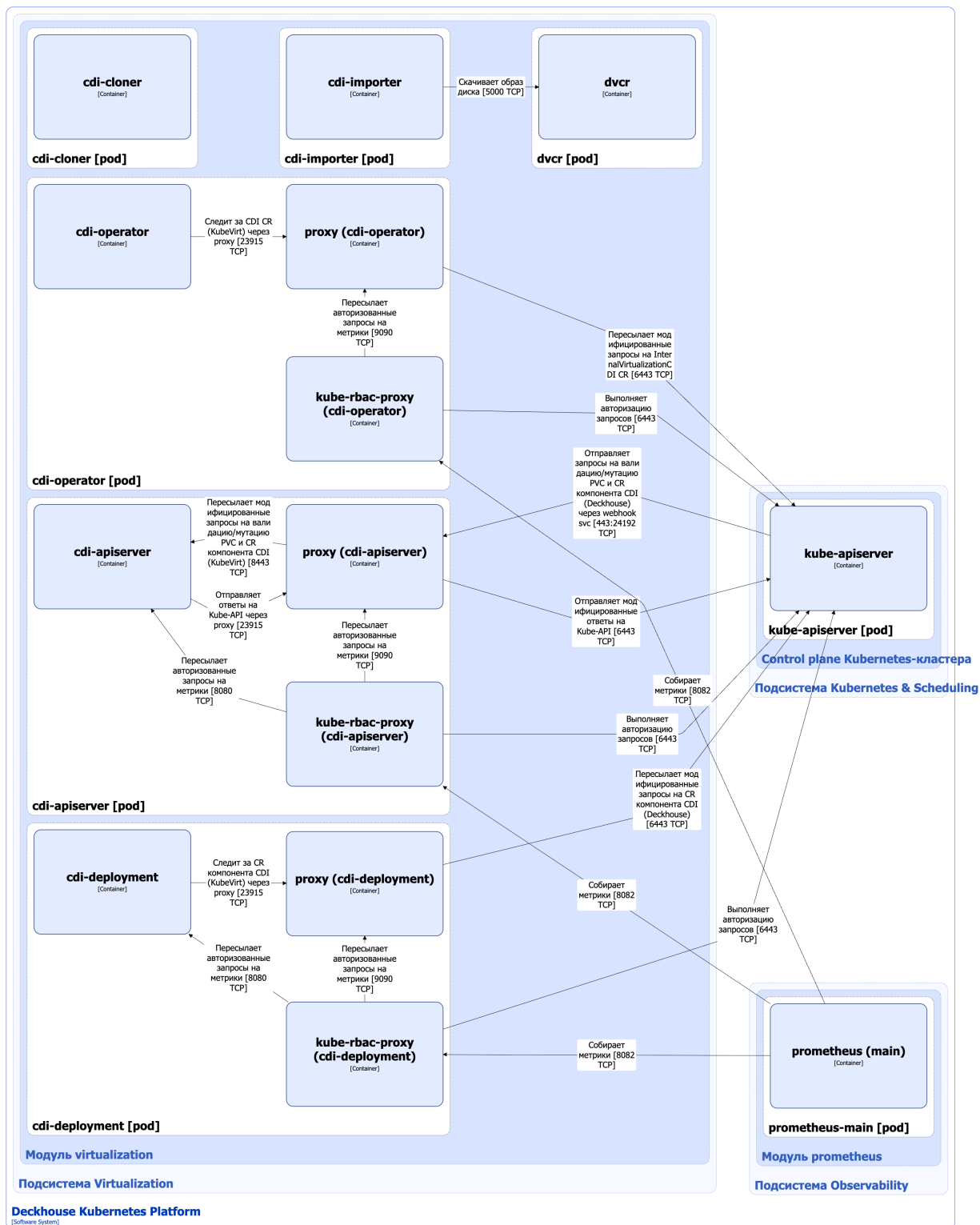
CDI использует кастомные ресурсы для управления дисками. Кастомный ресурс `InternalVirtualizationDataVolume` является абстракцией поверх стандартного PVC для Kubernetes и может использоваться для автоматизации создания и заполнения PVC данными.

Архитектура CDI

i Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура компонента CDI модуля `virtualization` на уровне 2 модели C4 и его взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты CDI

CDI состоит из следующих компонентов:

1. **Cdi-operator** — оператор Kubernetes, управляющий жизненным циклом компонентов CDI при помощи кастомного ресурса `InternalVirtualizationCDI`. **Cdi-operator** устанавливает в кластере **cdi-apiserver** и **cdi-deployment**, а также выполняет их настройку.

Компонент содержит следующие контейнеры:

- **cdi-operator** — основной контейнер;
- **proxy** (он же **kube-api-rewriter**) — сайдкар-контейнер, выполняющий модификацию проходящих через него запросов API, а именно переименование метаданных кастомных ресурсов. Это необходимо, поскольку компоненты KubeVirt используют API-группы вида `*.kubervirt.io`, а другие компоненты модуля `virtualization` используют аналогичные ресурсы, но с API-группами вида `*.virtualization.deckhouse.io`. Kube-api-rewriter является шлюзом, проксирующим

- запросы между контроллерами, управляющими ресурсами из разных API-групп;
 - **kube-rbac-proxy** — сайдкар-контейнер с авторизующим прокси на основе Kubernetes RBAC для организации защищенного доступа к метрикам контейнера проху. Является Open Source-проектом.
2. **Cdi-apiserver** — Kubernetes Extension API Server, который используется для проверки и изменения ресурсов Kubernetes API через механизмы Validating/Mutating Admission Controllers. Cdi-apiserver реализует validating- и mutating-вебхуки для следующих типов ресурсов:
- PersistentVolumeClaim — стандартный ресурс Kubernetes API;
 - InternalVirtualizationDataVolume — абстракция поверх стандартных PVC для создания на их основе дисков VM;
 - InternalVirtualizationCDI — кастомный ресурс, используемый cdi-operator, для установки и настройки компонентов CDI;
 - InternalVirtualizationDataImportCron — определяет задание cron для импорта образов дисков в виде PVC;
 - VolumeImportSource — определяет источники для импорта дисков.

Компонент содержит следующие контейнеры:

- **cdi-apiserver** — основной контейнер;
 - **proxy** (он же **kube-api-rewriter**) — сайдкар-контейнер, выполняющий модификацию проходящих через него запросов API. Подробно описан выше;
 - **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам контейнеров cdi-apiserver и проху. Подробно описан выше.
3. **Cdi-deployment** (он же **cdi-controller**) — контроллер, выполняющий следующие операции с кастомными ресурсами InternalVirtualizationDataVolumes:
- импорт образов и дисков VM в PVC-тома для использования их в виртуальных машинах, управляемых KubeVirt;
 - клонирование дисков (импорт в PVC из других PVC-томов или снимков);
 - синхронизация PVC с соответствующими кастомными ресурсами InternalVirtualizationDataVolumes.

Для выполнения некоторых из перечисленных выше операций контроллер создает и запускает временные поды:

- **cdi-importer** — для импорта образов и дисков VM. Cdi-importer также конвертирует образы в зависимости от типа целевого PVC:
 - в формат `raw`, если для PVC установлен режим тома (volume mode) `Block`;
 - в формат `qcow2`, если установлен режим `Filesystem`.
- **cdi-cloner** — для клонирования дисков и снимков.

Компонент содержит следующие контейнеры:

- **cdi-deployment** — основной контейнер, собранный на основе cdi-controller;
- **proxy** (он же **kube-api-rewriter**) — подробно описан выше;
- **kube-rbac-proxy** — сайдкар-контейнер, обеспечивающий авторизованный доступа к метрикам контейнеров cdi-deployment и проху. Подробно описан выше.

Взаимодействия CDI

CDI взаимодействует со следующими компонентами:

1. **Kube-apiserver**:
 - следит за кастомными ресурсами InternalVirtualizationCDI и стандартными ресурсами PersistentVolumeClaim;
 - выполняет авторизацию запросов к метрикам.
2. **KubeVirt** — предоставляет возможность импортировать образы и диски VM в PVC-тома для использования в VM, управляемых KubeVirt.
3. **DVCR (Deckhouse Virtualization Container Registry)** — использует хранилище образов в качестве источника для импорта образов и дисков VM.

С CDI взаимодействуют следующие внешние компоненты:

1. **Kube-apiserver** — выполняет валидацию/мутацию кастомных ресурсов компонента CDI и стандартных ресурсов PersistentVolumeClaim.
2. **Prometheus-main** — собирает метрики компонентов CDI.

Дополнительные ресурсы

- Документация модуля `virtualization`

Вспомогательные компоненты

В модуле `virtualization` используются компоненты, реализующие следующие вспомогательные функции:

- аудит событий безопасности;
- проброс USB-устройств в виртуальные машины;
- обновление сетевых маршрутов;
- удаление ресурсов перед деактивацией модуля `virtualization`.

Аудит событий безопасности

С инструкцией по активации аудита событий безопасности модуля `virtualization` можно ознакомиться в документации модуля.

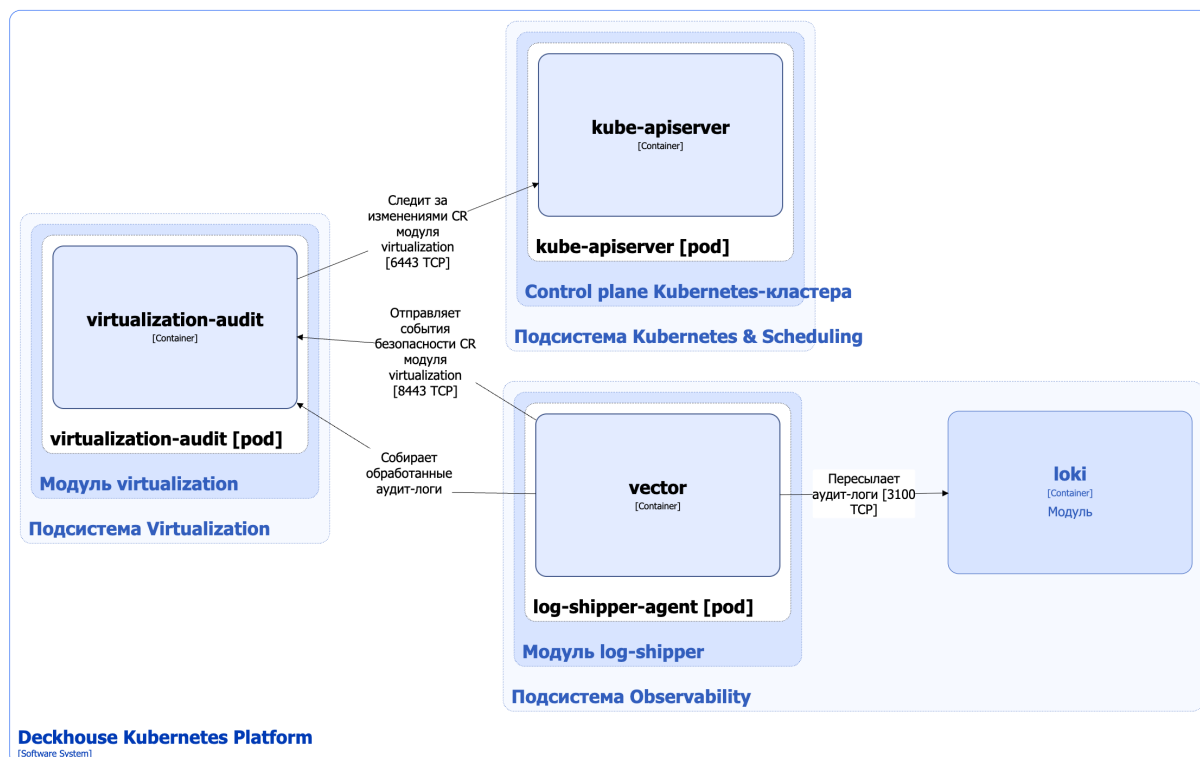
Архитектура

i

Для упрощения схемы приняты следующие допущения:

- На схеме контейнеры разных подов показаны как взаимодействующие напрямую. Фактически обмен выполняется через соответствующие сервисы Kubernetes (внутренние балансировщики). Названия сервисов не указываются, если они очевидны из контекста. В остальных случаях название сервиса приводится над стрелкой.
- Поды могут быть запущены в нескольких репликах, однако на схеме каждый под показан в единственном экземпляре.

Архитектура компонентов, реализующих аудит событий безопасности модуля `virtualization` на уровне 2 модели C4 и их взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты

Virtualization-audit — компонент, состоящий из одного контейнера и принимающий поток событий безопасности модуля `virtualization`. Отправка событий реализована с использованием модуля `log-shipper`. Агент логирования `vector` согласно настройкам в кастомных ресурсах `ClusterLoggingConfig` отбирает из аудит-лога кластера события, связанные с

кастомными ресурсами модуля `virtualization`, и отправляет их на эндпоинт сервиса `virtualization-audit`. `Virtualization-audit` обрабатывает полученные `audit`-события, обогащает их данными из `Kubernetes API` и сохраняет обработанные события в собственный лог.

Можно перенаправить события безопасности в систему логирования кластера (например, `Loki`). В этом случае аналогичным образом используются ресурсы `ClusterLoggingConfig` и агент `vector` модуля `log-shipper`.

Взаимодействия

`Virtualization-audit` взаимодействует со следующими компонентами:

1. **Kube-apiserver** — следит за изменениями кастомных ресурсов модуля `virtualization`.

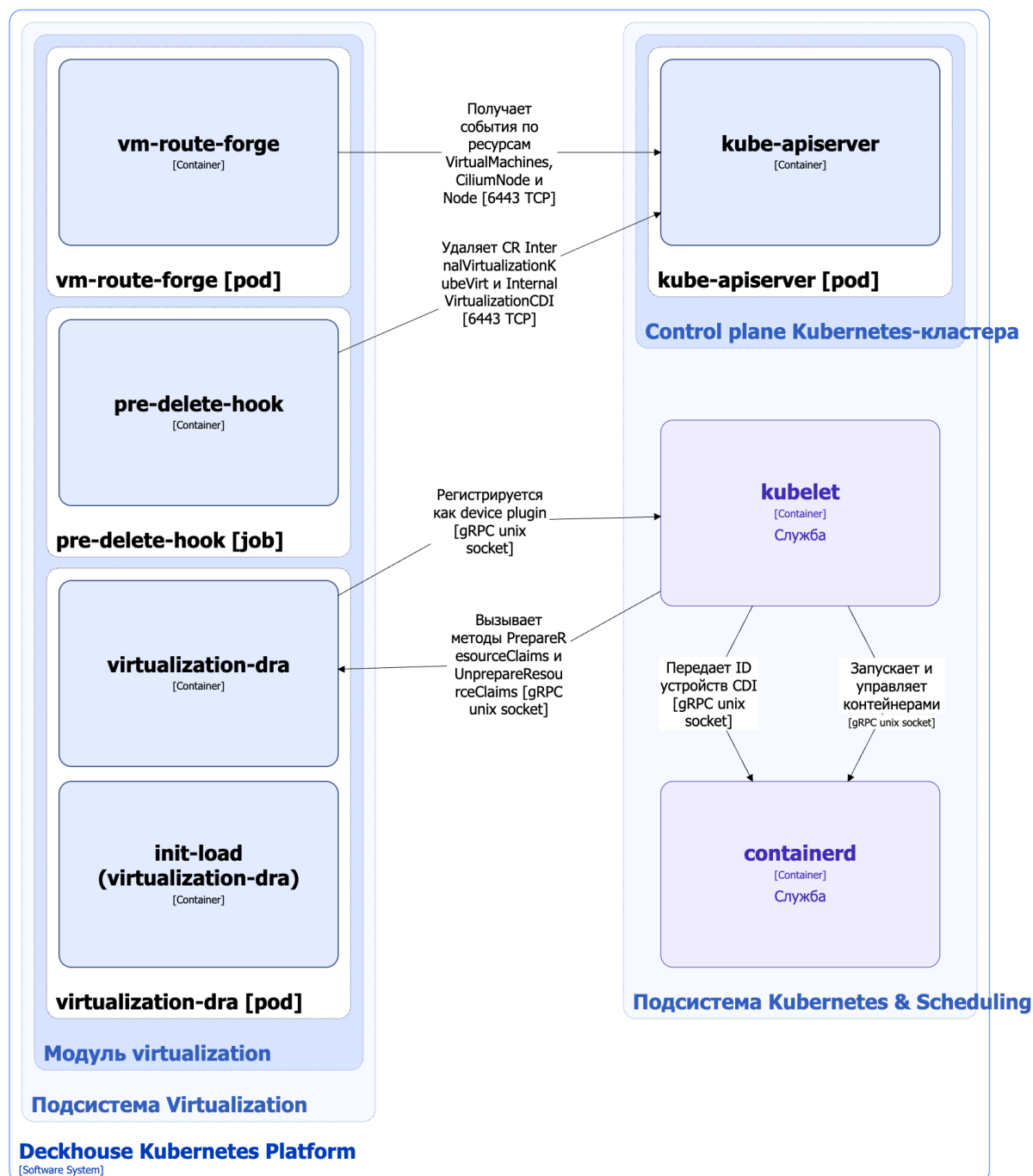
С `Virtualization-audit` взаимодействуют следующие внешние компоненты:

1. **Log-shipper-agent**:
 - отправляет события безопасности модуля `virtualization`;
 - собирает обработанные аудит-логи.

Virtualization-dra и прочие компоненты

Архитектура

Архитектура прочих вспомогательных компонентов модуля `virtualization` на уровне 2 модели C4 и их взаимодействия с другими компонентами DKP изображены на следующей диаграмме:



Компоненты

1. **Virtualization-dra** (DaemonSet) — драйвер DRA, с помощью которого реализуется проброс USB-устройств в виртуальные машины. Для проброса USB-устройств используется технология DRA (Dynamic Resource Allocation). DRA — это механизм Kubernetes API, scheduler и kubelet для описания, планирования и подготовки динамически выделяемых ресурсов через внешние драйверы. Драйвер DRA выполняет следующие операции:

- автоматически обнаруживает USB-устройства на узлах кластера и публикует их как ресурс ResourceSlice. virtualization-controller синхронизирует эти данные в кастомные ресурсы NodeUSBDevice, которые дальше используются для настройки проброса USB-устройств. Подробнее с настройкой проброса USB-устройств можно ознакомиться в документации модуля;
- регируется в kubelet как DRA kubelet plugin. DRA kubelet plugin подготавливает и освобождает выделенные ресурсы для подов через операции PrepareResourceClaims и UnprepareResourceClaims. Метод PrepareResourceClaims возвращает ID устройств CDI (Container Device Interface), которые kubelet передаёт в containerd. Данные о доступных USB-устройствах публикуются через ResourceSlice, а выбор устройств выполняется механизмом Kubernetes DRA на основе ResourceClaim/ResourceClaimTemplate и DeviceClass.

DRA-драйвер взаимодействует с kubelet по протоколу gRPC через Unix-сокеты.

- реализует USBIP-сервер, благодаря чему USB-устройство автоматически по сети пробрасывается на узел, где запущена виртуальная машина. Нет необходимости вручную размещать VM на том же узле, где находится устройство.

Состоит из следующих контейнеров:

- **init-load** — init-контейнер, загружающий модули ядра Linux, необходимые для работы DRA-драйвера;
 - **virtualization-dra** — основной контейнер.
2. **Vm-route-forge** — контроллер, следящий за кастомными ресурсами VirtualMachine API-группы `virtualization.deckhouse.io` и обновляющий сетевые маршруты на узле через Linux netlink/eBPF в таблицах маршрутизации, используемых CNI Cilium для маршрутизации трафика между VM.
 3. **Pre-delete-hook** (Job) — задача, запускаемая контроллером Deckhouse перед удалением модуля `virtualization`. Она удаляет кастомные ресурсы `InternalVirtualizationKubeVirt` и `InternalVirtualizationCDI` с именем `config`.

Взаимодействия

Virtualization-dra взаимодействует со следующими компонентами:

1. **Kubelet** — регистрируется в kubelet как DRA kubelet plugin.

Vm-route-forge взаимодействует со следующими компонентами:

1. **Kube-apiserver** — получает события по ресурсам VirtualMachine, CiliumNode и Node.
2. **Сеть хоста/ядро Linux** — обновляет маршруты и правила маршрутизации на узле.
3. **Cilium data plane** — использует данные CiliumNode и таблиц маршрутизации Cilium для маршрутизации трафика VM.

Pre-delete-hook взаимодействует со следующими компонентами:

1. **Kube-apiserver** — удаляет ресурсы `InternalVirtualizationKubeVirt` и `InternalVirtualizationCDI` с именем `config`.

C Virtualization-dra взаимодействуют следующие внешние компоненты:

1. **Kubelet** — вызывает gRPC-методы `PrepareResourceClaims` и `UnprepareResourceClaims` для подготовки и освобождения ресурсов, связанных с USB-устройствами.

Дополнительные ресурсы

- Документация модуля `cni-cilium`
- Документация модуля `log-shipper`
- Документация модуля `virtualization`

Глобальные настройки

Глобальные настройки Deckhouse Kubernetes Platform позволяют вам настраивать параметры, которые используются по умолчанию всеми модулями и компонентами. Некоторые модули могут переопределять часть этих параметров (это можно узнать в разделе настройки соответствующего модуля в документации модуля).

Глобальные настройки Deckhouse хранятся в ModuleConfig `global`.

i

В параметре `publicDomainTemplate` указывается шаблон DNS-имен, с учётом которого некоторые модули Deckhouse создают Ingress-ресурсы. Если параметр не указан, Ingress-ресурсы создаваться не будут.

Если у вас нет возможности заводить wildcard-записи DNS, для тестирования можно воспользоваться сервисом `sslip.io` или его аналогами.

Домен, указанный в шаблоне, не может совпадать или быть поддоменом домена, заданного в параметре `clusterDomain`. Мы не рекомендуем менять значение `clusterDomain` без особой необходимости.

Для корректной работы шаблона необходимо предварительно настроить службы DNS как в сетях, где будут располагаться узлы кластера, так и в сетях, из которых к служебным веб-интерфейсам платформы будут обращаться клиенты.

В случае, если шаблон совпадает с доменом сети узлов, используйте только A записи для назначения служебным веб-интерфейсам платформы адресов Frontend узлов. Например, для узлов заведена зона `company.my`, а шаблон имеет вид `%s.company.my`.

Пример ресурса `ModuleConfig/global`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings: # <-- Параметры модуля из раздела "Параметры" ниже.
    defaultClusterStorageClass: 'default-fast'
  modules:
    publicDomainTemplate: '%s.kube.company.my'
    resourcesRequests:
      controlPlane:
        cpu: 1000m
        memory: 500M
    placement:
      customTolerationKeys:
        - dedicated.example.com
    storageClass: 'default-fast'
```

Конверсии

Глобальные параметры настраиваются с помощью `ModuleConfig global`, схема которого содержит номер версии. При применении в кластере *старой* версии схемы `ModuleConfig` выполняются автоматические преобразования. Чтобы обновить версию схемы `ModuleConfig` вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Перенесите `storageClass` в `modules.storageClass`. Удалите параметры `modules.resourcesRequests.everyNode`, `modules.resourcesRequests.masterNode` и `modules.proxy`.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.defaultClusterStorageClass**
строка

Имя `StorageClass`, который DKP определит как `StorageClass` используемый в кластере по умолчанию.

DKP запрещает вручную добавлять или изменять default-аннотацию (`storageclass.kubernetes.io/is-default-class='true'`) у `StorageClass`. Используйте

параметр, чтобы определять или изменять StorageClass используемый в кластере по умолчанию.

Если значение не указано, фактический StorageClass используемый в кластере по умолчанию будет определяться в следующем порядке:

- StorageClass в кластере, имеющий default-аннотацию (`storageclass.kubernetes.io/is-default-class='true'`);
- Первый (по алфавиту) StorageClass из создаваемых модулем облачного провайдера (если какой-либо модуль облачного провайдера используется).

По умолчанию:

◦ **settings.highAvailability**
булевый

Параметр глобального включения *режима отказоустойчивости* для модулей, которые его поддерживают.

Если значение не указано, оно определяется автоматически как `true` для кластеров с более чем одним master-узлом. В остальных случаях значение определяется как `false`.

Примеры:

```
highAvailability: true
```

```
highAvailability: false
```

◦ **settings.modules**
объект

Общие параметры модулей Deckhouse.

■ **settings.modules.gatewayAPIGateway**
объект

Ссылка на объект Gateway (из Gateway API), используемый модулями экосистемы Deckhouse при создании ресурсов Gateway API.

Если параметр не указан, модули используют Gateway, автоматически обнаруженный в кластере (если он есть).

■ **settings.modules.gatewayAPIGateway.name**
строка

Обязательный параметр

Имя объекта Gateway.

■ **settings.modules.gatewayAPIGateway.namespace**
строка

Обязательный параметр

Неймспейс объекта Gateway.

■ **settings.modules.https**
объект

Способ реализации протокола HTTPS, используемый модулями Deckhouse.

Примеры:

```
https:
  certManager:
    clusterIssuerName: letsencrypt
  mode: CertManager
```

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: plainstring
```

■ **settings.modules.https.certManager**

объект

■ **settings.modules.https.certManager.clusterIssuerName**

строка

Имя ClusterIssuer, используемого модулями для получения TLS-сертификатов.

Модуль `cert-manager` предоставляет следующие ClusterIssuer:

- `letsencrypt`;
- `letsencrypt-staging`;
- `selfsigned`;
- `cloudflare`;
- `digitalocean`;
- `route53`.

При необходимости вы можете использовать собственный ClusterIssuer.

По умолчанию: `letsencrypt`

■ **settings.modules.https.customCertificate**

объект

■ **settings.modules.https.customCertificate.secretName**

строка

Имя Secret'a в пространстве имен `d8-system`, который будет использоваться модулями Deckhouse.

Указанный Secret должен быть в формате `kubernetes.io/tls`.

По умолчанию: `false`

■ **settings.modules.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — модули Deckhouse будут работать по HTTPS, самостоятельно заказывая сертификат с помощью ClusterIssuer, указанного в параметре `certManager.clusterIssuerName`;
- `CustomCertificate` — модули Deckhouse будут работать по HTTPS, используя сертификат из пространства имен `d8-system`;
- `Disabled` — модули Deckhouse будут работать только по HTTP (некоторые модули могут не работать, например `user-authn`);
- `OnlyInURI` — модули Deckhouse будут работать по HTTP, подразумевая, что перед ними стоит внешний HTTPS-балансировщик, который терминирует HTTPS. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `CertManager`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

■ **settings.modules.ingressClass**

строка

Имя класса Ingress-контроллера (IngressClass), используемого модулями экосистемы Deckhouse по умолчанию при создании Ingress-ресурсов.

Некоторые модули поддерживают собственный параметр `ingressClass`, позволяющий использовать другой класс Ingress-контроллера вместо используемого по умолчанию.

По умолчанию: `nginx`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
ingressClass: nginx
```

■ **settings.modules.placement**

объект

Настройки, определяющие расположение компонентов модулей Deckhouse по умолчанию.

■ **settings.modules.placement.customTolerationKeys**

массив строк

Список ключей пользовательских toleration.

Необходимо указывать, чтобы позволить планировщику размещать критически важные компоненты Deckhouse, такие как cni и csi, на выделенных узлах.

Пример:

```
customTolerationKeys:
- dedicated.example.com
- node-dedicated.example.com/master
```

■ **settings.modules.publicDomainTemplate**

строка

Шаблон DNS-имен с ключом `%s` в качестве динамической части строки.

Модули Deckhouse используют этот шаблон при создании Ingress-ресурсов.

Например, для шаблона `%s.kube.company.my` модуль prometheus создаст Ingress-ресурс на адрес `grafana.kube.company.my` для доступа к Grafana.

Обратите внимание:

- Если параметр не указан, Ingress-ресурсы создаваться не будут.
- Домен, указанный в шаблоне, не может совпадать или быть поддоменом домена, заданного в параметре `clusterDomain`. Мы не рекомендуем менять значение `clusterDomain` без особой необходимости.
- Для корректной работы шаблона необходимо предварительно настроить службы DNS как в сетях, где будут располагаться узлы кластера, так и в сетях, из которых к служебным веб-интерфейсам платформы будут обращаться клиенты.
- В случае, если шаблон совпадает с доменом сети узлов, используйте только A записи для назначения служебным веб-интерфейсам платформы адресов Frontend узлов. Например, для узлов заведена зона `company.my`, а шаблон имеет вид `%s.company.my`.

Шаблон: `^(%s([-a-z0-9]*[a-z0-9])?|[-a-z0-9]([-a-z0-9]*)?%s([-a-z0-9]*)?[a-z0-9]|[-a-z0-9]([-a-z0-9]*)?%s)(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Примеры:

```
publicDomainTemplate: "%s.kube.company.my"
```

```
publicDomainTemplate: kube-%s.company.my
```

■ **settings.modules.resourcesRequests**

объект

Количество ресурсов (CPU и памяти), выделяемых для работы некоторых компонентов.

Подробнее про единицы измерения ресурсов.

■ **settings.modules.resourcesRequests.controlPlane**

объект

Объём ресурсов (CPU и памяти), выделяемых для компонентов control plane на каждом master-узле кластера.

Настройки не применяются, если control plane кластера управляется облачным провайдером (например, Google Kubernetes Engine или Azure Kubernetes Service).

Пример:

```
controlPlane:
  cpu: 1000m
  memory: 500M
```

- **settings.modules.resourcesRequests.controlPlane.cpu**

строка от число

Количество CPU, выделяемых для компонентов control plane, работающих на каждом master-узле кластера.

Шаблон: `^[0-9]+m?$`

- **settings.modules.resourcesRequests.controlPlane.memory**

строка

Объём памяти, выделяемой для компонентов control plane, работающих на каждом master-узле кластера.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.modules.storageClass**

строка

Имя StorageClass, который будет использоваться для всех компонентов Deckhouse Kubernetes Platform (Prometheus, Grafana, OpenVPN и т. д.).

- Если значение не указано, то используется StorageClass по умолчанию, установленный в кластере.
- Этот параметр имеет смысл использовать только в исключительных ситуациях.
- Этот параметр применяется в процессе включения модуля.

По умолчанию:

- **settings.storageClass**

Параметр устарел
строка



Устарел. Используйте параметр settings.modules.storageClass.

Имя StorageClass, который будет использоваться для всех компонентов Deckhouse Kubernetes Platform (Prometheus, Grafana, OpenVPN и т. д.).

- Если значение не указано, то используется StorageClass по умолчанию, установленный в кластере.
- Этот параметр имеет смысл использовать только в исключительных ситуациях.
- Этот параметр применяется в процессе включения модуля.

По умолчанию:

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Custom Resources

Application

Scope: Namespaced
Version: v1alpha1

Представляет экземпляр приложения в пространстве имён.

- **apiVersion**
строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации Kubernetes.

- **kind**
строка

Значение представляет REST-ресурс, который описывает данный объект. Серверы могут определить его по эндпоинту, к которому клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации Kubernetes.

- **metadata**
объект
- **spec**
объект

Обязательный параметр

Определяет конфигурацию приложения.

- **spec.packageName**
строка

Обязательный параметр

Название пакета приложения для установки.

- **spec.packageRepositoryName**
строка

Название репозитория, где находится пакет. Если не указано, используется репозиторий по умолчанию.

- **spec.packageVersion**
строка

Обязательный параметр

Версия пакета приложения для установки.

- **spec.releaseChannel**
строка

Канал релиза для пакета приложения.

- **spec.settings**
объект

Настройки конфигурации для приложения.

- **status**
объект

Статус приложения.

- **status.conditions**
массив объектов

Условия, отражающие последние наблюдения за состоянием приложения.

Условие, описывающее один аспект текущего состояния приложения.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

Последнее время перехода условия из одного статуса в другой.

- **status.conditions.message**
строка

Обязательный параметр

Человеко-читаемое сообщение, указывающее детали перехода.

Максимальная длина: 32768

- **status.conditions.observedGeneration**

целочисленный

Значение `.metadata.generation`, на котором было установлено условие. Позволяет определить актуальность условия для текущего состояния ресурса.

Допустимые значения: `0 <= X`

- **status.conditions.reason**

строка

Обязательный параметр

Программный идентификатор причины последнего перехода условия.

Шаблон: `^[A-Za-z]([A-Za-z0-9_]*[A-Za-z0-9_])?$`

Длина: `1..1024`

- **status.conditions.status**

строка

Обязательный параметр

Статус условия. Может быть `True`, `False` или `Unknown`.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**

строка

Обязательный параметр

Тип условия приложения в CamelCase или в формате `foo.example.com/CamelCase`.

Шаблон: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*\/)?(((A-Za-z0-9)[:-A-Za-z0-9_]*)?[A-Za-z0-9])$`

Максимальная длина: `316`

- **status.currentVersion**

объект

Информация о текущей установленной версии.

- **status.currentVersion.channel**

строка

Канал релиза, из которого была установлена версия.

- **status.currentVersion.version**

строка

Семантическая версия установленного приложения.

- **status.lastAppliedConfiguration**

объект

Конфигурация (пользовательские настройки с учётом значений по умолчанию из схемы конфигурации), на основе которой было выполнено последнее успешное применение.

- **status.summary**

объект

Агрегированная сводка состояния приложения, сообщения и подсказки для пользователя. Контроллер заполняет её при каждой синхронизации — каждое приложение соответствует ровно одному состоянию жизненного цикла; это единый источник истины для UI: клиентам не следует вычислять эти значения повторно из `conditions`.

- **status.summary.message**

строка

Человеко-читаемое описание текущего состояния.

- **status.summary.state**

строка

Высокоуровневое состояние жизненного цикла приложения. Всегда одно из: `Pending`, `Failed`, `Updating`, `Ready`, `Degraded`, `Suspended`.

- **status.summary.tip**

строка

Человеко-читаемая инструкция о том, как разрешить текущее состояние. Пустое значение, если действий не требуется.

- **status.tracking**

объект

Отображение Nelm-трекинга.

ApplicationPackage

Scope: Cluster
Version: v1alpha1

Представляет информацию о доступном пакете приложения.

- **apiVersion**
строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации Kubernetes.

- **kind**
строка

Значение представляет REST-ресурс, который описывает данный объект. Серверы могут определить его по эндпоинту, к которому клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации Kubernetes.

- **metadata**
объект
- **status**
объект

Статус пакета приложения.

- **status.availableRepositories**
массив строк

Список названий репозитория, где доступен этот пакет приложения.

- **status.usedBy**
массив объектов

Информация о приложениях, использующих этот пакет.

- **status.usedBy.name**
строка

Название экземпляра приложения.

- **status.usedBy.namespace**
строка

Неймспейс, в котором установлено приложение.

- **status.usedBy.version**
строка

Версия пакета, используемая этим приложением.

- **status.usedByCount**
целочисленный

Количество приложений, использующих этот пакет.

ApplicationPackageVersion

Short names: apv

Scope: Cluster
Version: v1alpha1

Представляет версию пакета приложения.

- **apiVersion**
строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации Kubernetes.

- **kind**
строка

Значение представляет REST-ресурс, который описывает данный объект. Серверы могут определить его по эндпоинту, к которому клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации Kubernetes.

- **metadata**
объект
- **spec**
объект

Определяет параметры версии пакета приложения.

- **spec.packageName**

строка

Название пакета приложения.

- **spec.packageRepositoryName**

строка

Название репозитория пакетов, содержащего данный пакет.

- **spec.packageVersion**

строка

Версия пакета приложения.

- **status**

объект

Статус версии пакета приложения.

- **status.conditions**

массив объектов

Условия, отражающие последние наблюдения за состоянием версии пакета.

Условие, описывающее один аспект текущего состояния версии пакета.

- **status.conditions.lastTransitionTime**

строка

Обязательный параметр

Последнее время перехода условия из одного статуса в другой.

- **status.conditions.message**

строка

Обязательный параметр

Сообщение с деталями перехода. Может быть пустым.

Максимальная длина: 32768

- **status.conditions.observedGeneration**

целочисленный

Значение `.metadata.generation`, на котором было установлено условие. Позволяет определить актуальность условия для текущего состояния ресурса.

Допустимые значения: 0 <= X

- **status.conditions.reason**

строка

Обязательный параметр

Программный идентификатор причины последнего перехода условия.

Шаблон: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Длина: 1..1024

- **status.conditions.status**

строка

Обязательный параметр

Статус условия. Может быть `True`, `False` или `Unknown`.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**

строка

Обязательный параметр

Тип условия версии пакета в CamelCase или в формате `foo.example.com/CamelCase`.

Шаблон: `^([a-z0-9]([a-z0-9]*[a-z0-9])?(\.[a-z0-9]([a-z0-9]*[a-z0-9])?)*\/)?(((A-Za-z0-9)[:-A-Za-z0-9_])*)?[A-Za-z0-9]$`

Максимальная длина: 316

- **status.packageMetadata**

объект

Метаданные о пакете, такие как описание, требования и т.д.

- **status.packageMetadata.category**

строка

Категория, к которой принадлежит этот пакет.

- **status.packageMetadata.changelog**

объект

Информация об изменениях в этой версии.

- **status.packageMetadata.changelog.features**

массив строк

Список новых функций в этой версии.

- **status.packageMetadata.changelog.fixes**

массив строк

Список исправлений ошибок в этой версии.

- **status.packageMetadata.description**

объект

Локализованные описания пакета.

- **status.packageMetadata.description.en**

строка

Английское описание пакета.

- **status.packageMetadata.description.ru**

строка

Русское описание пакета.

- **status.packageMetadata.licensing**

объект

Информация о лицензировании для разных редакций.

- **status.packageMetadata.licensing.editions**

объект

Информация о лицензировании для разных редакций пакета.

- **status.packageMetadata.licensing.editions.<KEY_NAME>**

объект

<KEY_NAME> — имя параметра (ключ объекта).

- **status.packageMetadata.licensing.editions.<KEY_NAME>.available**

булевый

Доступна ли эта редакция для использования.

- **status.packageMetadata.requirements**

объект

Системные требования для этого пакета.

- **status.packageMetadata.requirements.deckhouse**

объект

Требуемая версия Deckhouse Kubernetes Platform .

- **status.packageMetadata.requirements.deckhouse.constraint**

строка

Выражение semver-ограничения.

- **status.packageMetadata.requirements.kubernetes**

объект

Требуемая версия Kubernetes.

- **status.packageMetadata.requirements.kubernetes.constraint**

строка

Выражение semver-ограничения.

- **status.packageMetadata.requirements.modules**

объект

Требуемые модули, разделённые на обязательные, условные, группы альтернатив и группы запрещённых модулей.

- **status.packageMetadata.requirements.modules.anyOf**

массив объектов

Группы альтернативных зависимостей — хотя бы один модуль из каждой группы должен быть установлен (и удовлетворять ограничению, если оно задано), чтобы пакет мог запуститься. Группы не добавляют рёбер в граф зависимостей.

Группа альтернативных зависимостей. Хотя бы один из перечисленных модулей должен быть установлен.

- **status.packageMetadata.requirements.modules.anyOf.description**
строка
Человеко-читаемое описание назначения группы.
- **status.packageMetadata.requirements.modules.anyOf.modules**
массив объектов
Обязательный параметр
Альтернативные зависимости в группе.
Именованная зависимость модуля с опциональным semver-ограничением. Пустое ограничение означает «любая версия».
 - **status.packageMetadata.requirements.modules.anyOf.modules.constraint**
строка
Выражение semver-ограничения.
 - **status.packageMetadata.requirements.modules.anyOf.modules.name**
строка
Обязательный параметр
Имя модуля.
- **status.packageMetadata.requirements.modules.anyOf.name**
строка
Обязательный параметр
Стабильный идентификатор группы, используемый в диагностических сообщениях планировщика.
- **status.packageMetadata.requirements.modules.conditional**
массив объектов
Условные зависимости — модуль не обязан быть установлен, но если он установлен, его версия должна удовлетворять ограничению, чтобы пакет работал корректно. Заменяет устаревший суффикс `!optional` из формата требований v1.
Именованная зависимость модуля с опциональным semver-ограничением. Пустое ограничение означает «любая версия».
 - **status.packageMetadata.requirements.modules.conditional.constraint**
строка
Выражение semver-ограничения.
 - **status.packageMetadata.requirements.modules.conditional.name**
строка
Обязательный параметр
Имя модуля.
- **status.packageMetadata.requirements.modules.mandatory**
массив объектов
Обязательные зависимости — модуль должен быть установлен (и удовлетворять ограничению версии, если оно задано), чтобы пакет мог запуститься.
Именованная зависимость модуля с опциональным semver-ограничением. Пустое ограничение означает «любая версия».
 - **status.packageMetadata.requirements.modules.mandatory.constraint**
строка
Выражение semver-ограничения.
 - **status.packageMetadata.requirements.modules.mandatory.name**
строка
Обязательный параметр
Имя модуля.
- **status.packageMetadata.requirements.modules.noneOf**
массив объектов
Группы запрещённых зависимостей — ни один модуль из каждой группы не должен быть установлен, чтобы пакет мог запуститься. Член без ограничения запрещён в любой версии; член с ограничением запрещён только в версиях, удовлетворяющих ограничению. Группы не добавляют рёбер в граф зависимостей.
Группа запрещённых зависимостей. Ни один из перечисленных модулей не должен быть установлен.
 - **status.packageMetadata.requirements.modules.noneOf.description**
строка

Человеко-читаемое описание назначения группы.

- **status.packageMetadata.requirements.modules.noneOf.modules**
массив объектов

Обязательный параметр

Запрещённые зависимости в группе.

Именованная зависимость модуля с опциональным semver-ограничением. Пустое ограничение означает «модуль запрещён в любой версии».

- **status.packageMetadata.requirements.modules.noneOf.modules.constraint**
строка

Выражение semver-ограничения для запрещённого диапазона версий.

- **status.packageMetadata.requirements.modules.noneOf.modules.name**
строка

Обязательный параметр

Имя модуля.

- **status.packageMetadata.requirements.modules.noneOf.name**
строка

Обязательный параметр

Стабильный идентификатор группы, используемый в диагностических сообщениях планировщика.

- **status.packageMetadata.stage**
строка

Стадия разработки пакета (например, alpha, beta, stable).

- **status.packageMetadata.versionCompatibilityRules**
объект

Правила совместимости версий для обновлений и откатов.

- **status.packageMetadata.versionCompatibilityRules.downgrade**
объект

Правила совместимости для отката с этой версии.

- **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipMajor**
целочисленный

Сколько основных версий можно пропустить.

- **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipMinor**
целочисленный

Сколько второстепенных версий можно пропустить.

- **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipPatches**
целочисленный

Сколько патч-версий можно пропустить.

- **status.packageMetadata.versionCompatibilityRules.downgrade.from**
строка

Начальный диапазон версий для совместимости.

- **status.packageMetadata.versionCompatibilityRules.downgrade.maxRollback**
целочисленный

Максимальное количество версий, которые можно откатить.

- **status.packageMetadata.versionCompatibilityRules.downgrade.to**
строка

Конечный диапазон версий для совместимости.

- **status.packageMetadata.versionCompatibilityRules.upgrade**
объект

Правила совместимости для обновления до этой версии.

- **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipMajor**
целочисленный

Сколько основных версий можно пропустить.

- **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipMinor**
целочисленный

Сколько второстепенных версий можно пропустить.

- **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipPatches**
целочисленный
Сколько патч-версий можно пропустить.
- **status.packageMetadata.versionCompatibilityRules.upgrade.from**
строка
Начальный диапазон версий для совместимости.
- **status.packageMetadata.versionCompatibilityRules.upgrade.maxRollback**
целочисленный
Максимальное количество версий, которые можно откатить.
- **status.packageMetadata.versionCompatibilityRules.upgrade.to**
строка
Конечный диапазон версий для совместимости.
- **status.packageSchemas**
объект
Схемы OpenAPI для валидации настроек и значений, передаваемых пакету.
 - **status.packageSchemas.settingsSchema**
объект
Схема OpenAPI v3, используемая для валидации пользовательских настроек пакета (`Application.spec.settings`). Содержимое хранится как непрозрачный объект, поскольку представляет собой рекурсивную JSON-схему; её валидация выполняется контроллером при загрузке метаданных пакета.
 - **status.packageSchemas.valuesSchema**
объект
Схема OpenAPI v3, используемая для валидации итоговых значений (значения по умолчанию, объединённые с пользовательскими настройками), передаваемых в хуки и чарты пакета. Содержимое хранится как непрозрачный объект, поскольку представляет собой рекурсивную JSON-схему; её валидация выполняется контроллером при загрузке метаданных пакета.
- **status.usedBy**
массив объектов
Информация о приложениях, которые используют эту версию пакета.
 - **status.usedBy.name**
строка
Название экземпляра приложения.
 - **status.usedBy.namespace**
строка
Пространство имён, где установлено приложение.
- **status.usedByCount**
целочисленный
Количество приложений, использующих эту версию пакета.

ClusterConfiguration

Version: deckhouse.io/v1

Общие параметры кластера.

Определяют, например, сетевые параметры, параметры CRI, версию control plane и т. д. Некоторые параметры можно изменять после развёртывания кластера, во время его работы.

Чтобы изменить содержимое ресурса `ClusterConfiguration` в работающем кластере, выполните следующую команду:

```
d8 system edit cluster-configuration
```

Пример:

```

apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
podSubnetNodeCIDRPrefix: '24'
podSubnetCIDR: 10.244.0.0/16
serviceSubnetCIDR: 192.168.0.0/16
kubernetesVersion: '1.31'
clusterDomain: k8s.internal
clusterType: Cloud
cloud:
  prefix: k8s-dev
  provider: Yandex
proxy:
  httpProxy: https://user:password@proxy.company.my:8443
  httpsProxy: https://user:password@proxy.company.my:8443
  noProxy:
    - company.my

```

- **apiVersion**

строка

Обязательный параметр

Используемая версия API Deckhouse.

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **cloud**

объект

Параметры облачного провайдера (если используется `clusterType Cloud`).

- **cloud.prefix**

строка

Префикс объектов, создаваемых в облаке.

Используется, например, для того, чтобы отличать объекты, создаваемые для разных кластеров, для настройки маршрутизации и т. д.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **cloud.provider**

строка

Обязательный параметр

Облачный провайдер.

Допустимые значения: `OpenStack`, `AWS`, `GCP`, `Yandex`, `vSphere`, `Azure`, `VCD`, `Zvirt`, `Dynamix`, `Huaweicloud`, `DVP`

- **clusterDomain**

строка

Обязательный параметр

Домен кластера (используется для маршрутизации внутри кластера).

Обратите внимание: домен не должен совпадать с доменом, используемым в шаблоне DNS-имен в параметре `publicDomainTemplate`. Например, нельзя одновременно устанавливать `clusterDomain: cluster.local` и `publicDomainTemplate: %s.cluster.local`.



При необходимости смены параметра в уже развернутом кластере рекомендуется воспользоваться инструкцией

По умолчанию: `cluster.local`

- **clusterType**

строка

Обязательный параметр

Тип инфраструктуры кластера:

- **Static** — кластер на bare metal (физические серверы) или виртуальных машинах. В случае виртуальных машин подразумевается, что у Deckhouse нет доступа к API для управления виртуальными машинами (они управляются администратором с помощью обычных средств облачной инфраструктуры);
- **Cloud** — кластер, развернутый в облачной инфраструктуре. Этот тип подразумевает, что у Deckhouse есть доступ к API облачной инфраструктуры для управления виртуальными машинами.

Допустимые значения: `Cloud`, `Static`

- **defaultCRI**

строка

Тип container runtime, используемый на узлах кластера (в NodeGroup'ах) по умолчанию.

Если используется значение `NotManaged`, то Deckhouse не будет управлять (устанавливать и настраивать) container runtime. В этом случае образы, используемые в NodeGroup'ах, должны содержать уже установленный container runtime.

Если установлено значение `ContainerdV2`, будет использоваться `CgroupsV2` (обеспечивает улучшенную безопасность и управление ресурсами). Для использования `ContainerdV2` в качестве container runtime узлы кластера должны соответствовать следующим требованиям:

- поддержка `CgroupsV2`;
- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- systemd версии 244 и новее;
- поддержка модуля ядра `erofs`.

По умолчанию: `Containerd`

Допустимые значения: `Containerd`, `ContainerdV2`, `NotManaged`

- **encryptionAlgorithm**

строка

Начиная с версии **1.31**, kubeadm использует указанный алгоритм асимметричного шифрования при генерации ключей и сертификатов для следующих компонентов control-plane:

- `apiserver`
- `apiserver-kubelet-client`
- `apiserver-etcd-client`
- `front-proxy-client`
- `etcd-server`
- `etcd-peer`
- `etcd-healthcheck-client`

При изменении этого параметра, сертификаты для выше указанных компонентов будут перевыпущены с выбранным алгоритмом и длиной ключа.

i

Внимание. При перевыпуске сертификатов не происходит ротация корневого сертификата (CA). Корневой сертификат создаётся с выбранным алгоритмом только при первичном бутстрапе кластера.

По умолчанию: `RSA-2048`

Допустимые значения: `RSA-2048`, `RSA-3072`, `RSA-4096`, `ECDSA-P256`

- **kind**

строка

Обязательный параметр

Допустимые значения: `ClusterConfiguration`

- **kubernetesVersion**

строка

Обязательный параметр

Версия Kubernetes (компонентов control plane кластера).

Изменение параметра в работающем кластере приведет к автоматическому обновлению версии control plane кластера.

Если указано `Automatic`, то используется версия control plane, которая считается стабильной на данный момент. Если стабильная версия control plane меньше максимальной, которая когда либо была установлена в кластере, больше чем на 1 минорную версию - тогда версия кластера не изменится. Изменение этой версии может произойти при смене минорной версии релиза Deckhouse (см. список изменений к релизу).

Допустимые значения: `1.31`, `1.32`, `1.33`, `1.34`, `1.35`, `Automatic`

- **podSubnetCIDR**

строка

Обязательный параметр

Адресное пространство подов кластера.

i

Внимание. Изменение этого параметра в работающем кластере может привести к полному отказу кластера. Изменение заблокировано по умолчанию. Чтобы обойти эту защиту, используйте команду `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing`. Однако **настоятельно рекомендуется пересоздать кластер** вместо изменения этого параметра. Подробнее см. в документации.

- **podSubnetNodeCIDRPrefix**

строка

Размер префикса подсети подов, выделяемой каждому узлу.

Этот параметр определяет, сколько IP-адресов доступно для подов на каждом отдельном узле. Например, если `podSubnetCIDR: 10.111.0.0/16` и `podSubnetNodeCIDRPrefix: "24"`, каждый узел получит подсеть /24 (например, `10.111.0.0/24`, `10.111.1.0/24` и т.д.), предоставляя до 254 IP-адресов на узел (253 используются для Pod'ов с учётом сетевого и широковещательного адресов).

Примеры расчёта:

- При сети /16 и префиксе /24: максимум **256 узлов** ($2^{(24-16)} = 256$).
- При сети /16 и префиксе /25: максимум **512 узлов**, но только **126 IP на узел**.
- Каждый IP-адрес сети подов может поддерживать до 2 подов (один запущенный + один завершающийся).

i

Внимание. Изменение этого параметра в работающем кластере может привести к полному отказу кластера. Изменение заблокировано по умолчанию. Чтобы обойти эту защиту, используйте команду `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing`. Однако **настоятельно рекомендуется пересоздать кластер** вместо изменения этого параметра. Подробнее см. в документации.

По умолчанию: 24

- **проxy**
объект

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Глобальная настройка проxy-сервера (в основном для работы в закрытых окружениях).

Описанные в этой секции параметры будут транслированы в переменные окружения `HTTP_PROXY`, `HTTPS_PROXY` и `NO_PROXY` для всех узлов кластера и компонентов Deckhouse. Это приведёт к тому, что HTTP(S) запросы (например, при выполнении `curl`, `git` и т. д.) ко всем не указанным в параметре `noProxy` ресурсам будут выполняться через прокси. Подсети `podSubnetCIDR`, `serviceSubnetCIDR` и домен `clusterDomain` добавляются в `noProxy` автоматически.

i

Внимание. Чтобы избежать использования прокси в запросах между подами кластера и сервисами расположенными в сети узлов кластера, заполните параметр `noProxy` списком подсетей, которые используются на узлах.

- **проxy.httpProxy**
строка

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

URL проxy-сервера для HTTP-запросов.

При необходимости укажите имя пользователя, пароль и порт проxy-сервера.

Шаблон: `^https?:/([!*'();&=+$,/?%#\[\]\0-9a-zA-Z\.\-\_]+(\[:[!*'();&=+$,/?%#\[\]\0-9a-zA-Z\.\-\_]+\)?@[0-9a-zA-Z\.\-\_]+(\:[0-9]{1,5})?)?$`

Примеры:

```
httpProxy: http://proxy.company.my
```

```
httpProxy: https://user:password@proxy.company.my:8443
```

```
httpProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
```

```
httpProxy: https://user%40domain.local:password@proxy.company.my:8443
```

- **проxy.httpsProxy**
строка

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

URL проxy-сервера для HTTPS-запросов.

При необходимости укажите имя пользователя, пароль и порт проxy-сервера.

Шаблон: `^https?:/([!*'();&=+$,/?%#\[\]\0-9a-zA-Z\.\-\_]+(\[:[!*'();&=+$,/?%#\[\]\0-9a-zA-Z\.\-\_]+\)?@[0-9a-zA-Z\.\-\_]+(\:[0-9]{1,5})?)?$`

Примеры:

```
httpsProxy: http://proxy.company.my
```

```
httpsProxy: https://user:password@proxy.company.my:8443
```

```
httpsProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
```

```
httpsProxy: https://user%40domain.local:password@proxy.company.my:8443
```

- **proxy.noProxy**
массив строк

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Список IP и доменных имен, для которых проксирование не применяется.

Для настройки wildcard-доменов используйте написание вида “.example.com”.

i **Внимание.** Если в кластере предполагается взаимодействие подов с сервисами, размещенными в сети узлов кластера, то укажите список подсетей, которые используются на узлах.

- **Элемент массива**
строка

Шаблон: `^[a-z0-9\-\.\.\/]+$`

- **serviceSubnetCIDR**
строка

Обязательный параметр

Адресное пространство для сервисов кластера.

i **Внимание.** Изменение этого параметра в работающем кластере может привести к полному отказу кластера. Изменение заблокировано по умолчанию. Чтобы обойти эту защиту, используйте команду `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing`. Однако **настоятельно рекомендуется пересоздать кластер** вместо изменения этого параметра. Подробнее см. в документации.

CNIMigration

Short names: cnim

Scope: Cluster

Version: v1alpha1

Определяет конфигурацию для миграции CNI.

- **spec**
объект

Определяет желаемое состояние миграции CNI.

- **spec.targetCNI**
строка

Целевой CNI, на который выполняется переключение (например, cilium, flannel).

- **status**
объект

Определяет наблюдаемое состояние миграции CNI.

- **status.conditions**
массив объектов

Отражает состояние миграции в целом. Список условий перехода между этапами.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

- **status.conditions.message**
строка

Обязательный параметр

- **status.conditions.reason**
строка

Обязательный параметр

- **status.conditions.status**
строка

Обязательный параметр

- **status.conditions.type**
строка

Обязательный параметр

- **status.currentCNI**
строка

Обнаруженный CNI, с которого выполняется переключение.

- **status.failedSummary**
массив объектов

Детализация ошибок по узлам.

- **status.failedSummary.node**
строка
- **status.failedSummary.reason**
строка

- **status.nodesFailed**
целочисленный

Количество узлов, на которых произошла ошибка.

- **status.nodesSucceeded**
целочисленный

Количество узлов, успешно завершивших текущую фазу.

- **status.nodesTotal**
целочисленный

Общее количество узлов, участвующих в миграции.

- **status.phase**
строка

Текущий этап процесса миграции.

CNINodeMigration

Short names: `cninm`

Scope: Cluster

Version: v1alpha1

Определяет конфигурацию для миграции CNI на конкретном узле.

- **spec**
объект

Определяет желаемое состояние миграции CNI на узле (пустой объект, конфигурация берется из глобального ресурса).

- **status**
объект

Определяет наблюдаемое состояние миграции CNI на узле.

- **status.conditions**
массив объектов

Список условий, отражающих шаги, выполненные на узле.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

- **status.conditions.message**
строка

Обязательный параметр

- **status.conditions.reason**
строка

Обязательный параметр

- **status.conditions.status**
строка

Обязательный параметр

- **status.conditions.type**
строка

Обязательный параметр

- **status.phase**

строка

Текущая фаза миграции на узле.

DeckhouseRelease

Scope: Cluster

Version: v1alpha1

Определяет состояние и параметры применения конкретного релиза (версии) Deckhouse Kubernetes Platform (DKP) в кластере.

Объекты DeckhouseRelease создаются DKP автоматически, при обнаружении новых версий DKP на выбранном канале обновлений. Изменение DeckhouseRelease позволяет управлять процессом применения соответствующей версии DKP. Подробнее о настройке обновлений DKP можно почитать в документации.

Актуальные версии DKP и модулей по каналам обновлений можно узнать на releases.deckhouse.ru.

- **approved**

булевый

Разрешает или запрещает ручное обновление.

Используется только если в конфигурации модуля `deckhouse` установлен ручной режим обновлений (параметр `update.mode` установлен в `Manual`). Игнорируется, если установлен режим обновления `Auto` или `AutoPatch`.

Подробнее о ручном подтверждении обновлений можно почитать в документации.

По умолчанию: `false`

- **spec**

объект

Обязательный параметр

- **spec.applyAfter**

строка

Время, до которого отложено обновление, если релиз является частью `canary-release`. Если релиз ожидает применения, уточнить причину можно в поле `.status.message` объекта DeckhouseRelease.

- **spec.changelog**

объект

Структура, содержащая список изменений DKP (и модулей, входящих в состав DKP) данного релиза. Подробнее об истории изменений в релизах DKP можно почитать в разделе «Обновление».

- **spec.changelogLink**

строка

Ссылка на страницу со всеми изменениями данного релиза.

- **spec.disruptions**

Параметр устарел
массив строк

Изменения в релизе, которые могут привести к кратковременному простоем в работе системных компонентов.

- **spec.requirements**

объект

Структура, содержащая список требований для установки релиза. Используется ядром DKP. Если требования не выполняются, релиз может быть пропущен или заблокирован для установки.

Сообщения о невыполненных требованиях, можно найти в поле `.status.message` объекта DeckhouseRelease.

- **spec.version**

строка

Обязательный параметр

Версия DKP.

Пример:

```
version: v1.73.2
```

- **status**

объект

- **status.approved**

булевый

Статус готовности релиза к обновлению. Используется только для режима обновления `Manual` (`update.mode: Manual`).

- **status.message**

строка

Детальное сообщение об ошибке или статусе релиза.

- **status.phase**

строка

Текущий статус релиза.

Типичные значения:

- **Pending** — релиз создан, но еще не применен (ожидание ручного подтверждения, оконч обновлений, сапарту-release, минимального времени уведомления);
- **Deployed** — DKP переключился на версию релиза (обновление модулей и компонентов может продолжаться асинхронно);
- **Superseded** — релиз устарел и больше не используется;
- **Suspended** — релиз отменен (как правило, до применения);
- **Skipped** — релиз пропущен из-за невыполненных требований (**requirements**).

Подробнее о статусах релизов DKP можно почитать в документации модуля **deckhouse**.

Допустимые значения: **Pending**, **Deployed**, **Outdated**, **Suspended**, **Superseded**, **Skipped**

- **status.transitionTime**

строка

Время изменения статуса релиза.

InitConfiguration

Version: deckhouse.io/v1

Конфигурация Deckhouse, с которой он запустится после установки.

Пример:

```
apiVersion: deckhouse.io/v1
kind: InitConfiguration
deckhouse:
  imagesRepo: nexus.company.my/deckhouse/ee
  registryDockerCfg:
eyJhdXRocyI6IHsgIm5leHVzLmNvbXBhbnkubXkiOiB7InVzZXJyYWllIjoibmV4dXMtdXNlciIsInBhc3N3b3JkIjoibmV4dX
MtcEBzc3cwcmQilLCJhdXRoiJoiYm1WNGRYTXRkWE5sY2pwdVpYaDFjeTF3UUh0emR6QnlaQW89In19fQo=
  registryScheme: HTTPS
  registryCA: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

- **apiVersion**

строка

Обязательный параметр

Используемая версия API Deckhouse.

Допустимые значения: **deckhouse.io/v1**, **deckhouse.io/v1alpha1**

- **deckhouse**

Параметр устарел
объект

Обязательный параметр

Параметры, необходимые для установки Deckhouse.

- **deckhouse.devBranch**

Параметр устарел
строка

Параметр используется для нужд разработки. Будет заменен на аргумент CLI-команды.

- **deckhouse.imagesRepo**

строка

Адрес container registry с образами Deckhouse.

Укажите, если используется редакция Enterprise Edition Deckhouse или сторонний container registry (например, прокси-сервер в закрытом окружении).

Адрес учитывает используемую редакцию Deckhouse. Публичный адрес container registry для Deckhouse Enterprise Edition — **registry.deckhouse.io/deckhouse/ee**.

По умолчанию: **registry.deckhouse.io/deckhouse/ce**

Шаблон: `^[0-9a-zA-Z\.\-]+\(:[0-9]{1,5}\)?(\/[0-9a-zA-Z\.\-\/]+)?$`

- **deckhouse.registryCA**

строка

Корневой сертификат, которым можно проверить сертификат container registry при работе по HTTPS (если registry использует самоподписанные SSL-сертификаты).

- **deckhouse.registryDockerCfg**

строка

Строка авторизации из файла конфигурации Docker-клиента (в Linux обычно это `$HOME/.docker/config.json`) для доступа к стороннему container registry, закодированная в Base64.

Например, для доступа к container registry `registry.company.my` под пользователем `user` с паролем `P@ssw0rd` это будет `eyJhdXRocyI6eyJyZWdpc3RyeS5jb2lwYW55Lm15Ijp7ImF1dGgiOiJkWE5sY2pwUVFITnpkekj5WkFvPSJ9fX0K` (строка `{"auths":{"registry.company.my":{"auth":"dXNlcjpwQHNzdzByZAo="}}` в Base64).

По умолчанию:

Пример:

```
registryDockerCfg: eyJhdXRocyI6IHsgInJlZ2lzdHJ5LmRlY2tob3VzZS5pbyI6IHt9fX0=
```

- **deckhouse.registryScheme**

строка

Протокол доступа к container registry (HTTP или HTTPS).

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

- **kind**

строка

Обязательный параметр

Допустимые значения: `InitConfiguration`

Module

Scope: Cluster

Version: v1alpha1

Описывает состояние модуля в кластере. Объект `Module` будет создан автоматически после настройки источника модулей (`ModuleSource`) и успешной синхронизации.

- **properties**

объект

- **properties.accessibility**

объект

Настройки доступности модуля.

- **properties.accessibility.editions**

объект

Настройки работы модуля в редакциях Deckhouse.

- **properties.accessibility.editions.<KEY_NAME>**

объект

`<KEY_NAME>` — имя параметра (ключ объекта).

- **properties.accessibility.editions.<KEY_NAME>.available**

булевый

- **properties.accessibility.editions.<KEY_NAME>.enabledInBundles**

массив строк

- **properties.availableSources**

массив строк

Доступные источники для скачивания модуля.

- **properties.critical**

булевый

Является ли модуль критичным.

- **properties.disableOptions**

объект

Параметры защиты от отключения модуля.

- **properties.disableOptions.confirmation**

булевый

- **properties.disableOptions.message**

строка

- **properties.exclusiveGroup**
строка

Принадлежность модуля к группе, где может быть активен только один модуль.
- **properties.namespace**
строка

Пространство имен в котором работает модуль.
- **properties.releaseChannel**
строка

Релизный канал модуля.
- **properties.requirements**
объект

Зависимости модуля — набор условий, которые должны выполняться чтобы Deckhouse Kubernetes Platform (DKP) мог запустить модуль.
 - **properties.requirements.bootstrapped**
строка

Требование к статусу установки кластера (только для встроенных модулей DKP).
 - **properties.requirements.deckhouse**
строка

Требование к версии Deckhouse.
 - **properties.requirements.kubernetes**
строка

Требование к версии Kubernetes.
 - **properties.requirements.modules**
объект

Список других включенных модулей, которые необходимы для работы модуля.
- **properties.source**
строка

Источник, из которого скачан модуль (иначе пусто).
- **properties.stage**
строка

Текущая стадия жизненного цикла модуля.
- **properties.subsystems**
массив строк

Подсистемы в которые входит модуль.
- **properties.updatePolicy**
строка

Имя политики обновления модуля.
- **properties.version**
строка

Версия модуля.
- **properties.weight**
целочисленный

Вес (приоритет) модуля.
- **status**
объект
 - **status.conditions**
массив объектов
 - **status.conditions.lastProbeTime**
строка
 - **status.conditions.lastTransitionTime**
строка
 - **status.conditions.message**
строка
 - **status.conditions.reason**
строка
 - **status.conditions.status**
строка
 - **status.conditions.type**
строка
 - **status.hooksState**
строка

Hooks status report.

- **status.phase**
строка

Module phase.

Допустимые значения: `Unavailable`, `Available`, `Downloading`, `DownloadingError`, `Reconciling`, `Installing`, `HooksDisabled`, `WaitSyncTasks`, `Downloaded`, `Conflict`, `Ready`, `Error`

ModuleConfig

Short names: `mc`

Scope: Cluster

Version: `v1alpha1`

Определяет конфигурацию модуля Deckhouse Kubernetes Platform (параметры модуля). Название ресурса `ModuleConfig` должно соответствовать названию модуля (например, `control-plane-manager` для модуля `control-plane-manager`).

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-1
spec:
  enabled: true
  settings: {}
  version: 1
```

- **spec**
объект

Обязательный параметр

- **spec.enabled**
булевый

Включить или выключить модуль.

Пример:

```
enabled: false
```

- **spec.maintenance**
строка

Задаёт режим управления работой модуля.

- `NoResourceReconciliation` — автономный режим для разработки или отладки модуля.

В этом режиме:

- Изменения в конфигурации или хуках модуля не синхронизируются, в результате чего ресурсы не обновляются автоматически.
- Отключается мониторинг ресурсов модуля, в результате чего удалённые ресурсы не восстанавливаются.
- Ко всем ресурсам модуля применяется лейбл `maintenance: NoResourceReconciliation`.
- Срабатывает алерт `ModuleIsInMaintenanceMode`.

Допустимые значения: `NoResourceReconciliation`

Пример:

```
maintenance: NoResourceReconciliation
```

- **spec.settings**
объект

Настройки модуля.

- **spec.source**
строка

Источник, из которого будет скачан модуль.

- **spec.updatePolicy**
строка

Политика обновления модуля.

Пример:

```
updatePolicy: test-alpha
```

- **spec.version**

число

Версия схемы настроек модуля.

Пример:

```
version: 1
```

- **status**

объект

- **status.message**

строка

Additional information

- **status.version**

строка

Version of settings schema in use

ModuleDocumentation

Scope: Cluster

Version: v1alpha1

Определяет конфигурацию рендеринга документации модуля Deckhouse.

Ресурсы ModuleDocumentation создает Deckhouse.

- **spec**

объект

Обязательный параметр

- **spec.checksum**

строка

Контрольная сумма версии модуля.

- **spec.path**

строка

Путь к версии модуля.

- **spec.version**

строка

Обязательный параметр

Версия модуля.

Пример:

```
version: v1.0.0
```

- **status**

объект

- **status.conditions**

массив объектов

- **status.conditions.address**

строка

- **status.conditions.checksum**

строка

- **status.conditions.lastTransitionTime**

строка

- **status.conditions.message**

строка

- **status.conditions.type**

строка

- **status.conditions.version**

строка

- **status.result**

строка

ModulePullOverride

Short names: mpo

Scope: Cluster

- v1alpha2
- v1alpha1

Определяет конфигурацию ресурса для скачивания конкретных версий модулей Deckhouse.

i **Важно.** Ресурс предназначен только для использования в средах разработки и отладки. Его применение в production-кластерах не рекомендуется. Поддержка ресурса может быть исключена в следующих версиях Deckhouse Kubernetes Platform.

- **spec**
объект

Обязательный параметр

- **spec.imageTag**
строка

Обязательный параметр

Тег образа контейнера модуля, который будет скачан и установлен.

- **spec.rollback**
булевый

Определение необходимости восстановления модуля из релиза, если объект ModulePullOverride удалён.

По умолчанию: `false`

- **spec.scanInterval**
строка

Интервал сканирования для проверки изменения образа контейнера. Если образ изменился, модуль будет обновлен. Поддерживаемые единицы: `h`, `m`, `s`.

По умолчанию: `15s`

Шаблон: `^(\d+h)?(\d+m)?(\d+s)?$`

- **status**
объект

- **status.imageDigest**
строка

Контрольная сумма образа контейнера.

- **status.message**
строка

Details of the resource status.

- **status.updatedAt**
строка

Когда модуль был обновлен в последний раз.

- **status.weight**
целочисленный

Вес (приоритет) модуля.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Определяет конфигурацию ресурса для скачивания конкретных версий модулей Deckhouse.

- **spec**
объект

Обязательный параметр

- **spec.imageTag**
строка

Обязательный параметр

Тег образа контейнера модуля, который будет скачан и установлен.

- **spec.rollback**
булевый

Indicates whether the module release should be rollback after deleting ModulePullOverride.

По умолчанию: `false`

- **spec.scanInterval**
строка

Интервал сканирования для проверки изменения образа контейнера. Если образ изменился, модуль будет обновлен. Поддерживаемые единицы: `h`, `m`, `s`.

По умолчанию: `15s`

Шаблон: `^(\d+h)?(\d+m)?(\d+s)?$`

- **spec.source**
строка

Обязательный параметр

Ссылка на объект ModuleSource модуля.
- **status**
объект
 - **status.imageDigest**
строка

Контрольная сумма образа контейнера.
 - **status.message**
строка

Details of the resource status.
 - **status.updatedAt**
строка

Когда модуль был обновлен в последний раз.
 - **status.weight**
целочисленный

Вес (приоритет) модуля.

ModuleRelease

Short names: mr

Scope: Cluster
Version: v1alpha1

Определяет конфигурацию релизов модулей Deckhouse.

Ресурсы ModuleRelease создаёт Deckhouse.

- **spec**
объект

Обязательный параметр
 - **spec.applyAfter**
строка

Время, до которого отложено обновление.
 - **spec.changelog**
объект

Список изменений модуля в данном релизе.
 - **spec.moduleName**
строка

Обязательный параметр

Название модуля.
 - **spec.requirements**
объект

Зависимости релиза — набор условий, которые должны выполняться чтобы Deckhouse Kubernetes Platform мог запустить эту версию модуля.
 - **spec.requirements.deckhouse**
строка

Требование к версии Deckhouse.
 - **spec.requirements.kubernetes**
строка

Требование к версии Kubernetes.
 - **spec.requirements.modules**
объект

Список других модулей, которые необходимы для этой версии модуля (должны быть включены).
 - **spec.update**
объект

Необязательные параметры обновления.
 - **spec.update.versions**

массив объектов

Список правил перехода `from` → `to`, позволяющих пропускать пошаговые обновления. Если текущая установленная версия модуля (статус `Deployed`) не ниже `from`, и в кластере есть релиз, версия которого соответствует `to`, контроллер пропустит промежуточные релизы и обновит модуль до версии из `to`. Значение `to` может задавать минорную ветку (`X.Y` — будет выбрана последняя доступная `X.Y.Z`). Правило указывается в самом релизе назначения (`constrained release`) — в том, версия которого соответствует `to`.

- **spec.update.versions.from**

строка

Обязательный параметр

Минимальная версия, с которой допускается переход (формат `X.Y`).

- **spec.update.versions.to**

строка

Обязательный параметр

Конечная версия диапазона — минорная ветка (`X.Y`).

- **spec.version**

строка

Обязательный параметр

Версия модуля в данном релизе.

Пример:

```
version: v1.0.0
```

- **spec.weight**

целочисленный

Вес (приоритет) модуля.

- **status**

объект

- **status.approved**

булевый

Статус готовности релиза к обновлению. Используется только для режима обновления `Manual` (`update.mode: Manual`).

- **status.message**

строка

Детальное сообщение об ошибке или статусе релиза.

- **status.phase**

строка

Текущий статус релиза.

Допустимые значения: `Pending`, `Deployed`, `Superseded`, `Suspended`, `Skipped`, `Terminating`

- **status.pullDuration**

строка

Время загрузки модуля.

- **status.size**

целочисленный

Размер образа модуля.

- **status.transitionTime**

строка

Время изменения статуса релиза.

ModuleSettingsDefinition

Scope: Cluster

Version: v1alpha1

Отображает настройки модулей Deckhouse. Определяет список версий настроек модуля.

- **spec**

объект

Обязательный параметр

Спецификация настроек модуля.

- **spec.versions**

массив объектов

Список версий настроек модуля. Каждая версия включает имя и схему.

- **spec.versions.conversions**
массив объектов

Список правил конверсий для данной версии.

Одно правило конверсии с выражениями и описаниями.
- **spec.versions.conversions.descriptions**
объект

Локализованные описания конверсии.
 - **spec.versions.conversions.descriptions.en**
строка

Английское описание конверсии.
 - **spec.versions.conversions.descriptions.ru**
строка

Русское описание конверсии.
- **spec.versions.conversions.expr**
массив строк

Массив jq выражений для преобразования настроек.
- **spec.versions.name**
строка

Обязательный параметр

Версия настроек модуля.
- **spec.versions.schema**
объект

Схема настроек для данной версии модуля.

ModuleSource

Short names: `ms`

Scope: Cluster
Version: v1alpha1

Определяет конфигурацию источника модулей Deckhouse.

Подробнее об установке модуля из источника — в разделе «Запуск модуля в кластере DKP»

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.io/modules-source
    dockerCfg: "<base64 encoded credentials>"
```

- **spec**
объект

Обязательный параметр
 - **spec.registry**
объект

Обязательный параметр
 - **spec.registry.ca**
строка

Корневой сертификат (В формате PEM), которым можно проверить сертификат registry при работе по HTTPS (если registry использует самоподписанные SSL-сертификаты).



Создание ресурса ModuleSource с указанием корневого сертификата, приведет к перезапуску containerd на всех узлах кластера.

- **spec.registry.dockerCfg**
строка

Строка с токеном доступа к container registry в Base64. В случае использования анонимного

доступа к container registry не заполняйте это поле.

- **spec.registry.repo**

строка

Обязательный параметр

Адрес репозитория образов контейнеров.

Пример:

```
repo: registry.example.io/deckhouse/modules
```

- **spec.registry.scheme**

строка

Протокол для доступа к репозиторию контейнеров.

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

- **spec.releaseChannel**

Параметр устарел
строка

Желаемый канал обновлений по умолчанию для модулей данного container registry.

- **spec.scanInterval**

строка

Интервал для сканирования container registry.

Определяет частоту проверки container registry на появление новых модулей и их версий.

По умолчанию: `3m`

Шаблон: `^(\d+h)?(\d+m)?(\d+s)?$`

Примеры:

```
scanInterval: 5m
```

```
scanInterval: 1h
```

```
scanInterval: 6h30m
```

- **status**

объект

- **status.message**

строка

- **status.modules**

массив объектов

Список доступных для установки модулей и их политики обновления.

- **status.modules.checksum**

строка

Чексумма релиза.

- **status.modules.error**

строка

Ошибка процессинга модуля.

- **status.modules.name**

строка

Имя модуля.

- **status.modules.overridden**

булевый

Показывает, есть ли ресурс `ModulePullOverride` для данного модуля.

- **status.modules.policy**

строка

Имя политики.

- **status.modules.pullError**

строка

Ошибка скачивания модуля.

- **status.modules.version**

строка

Версия модуля

- **status.modulesCount**
целочисленный

Количество доступных модулей.

- **status.phase**
строка

Текущая фаза.

Допустимые значения: `Active`, `Terminating`

- **status.syncTime**
строка

Время последней синхронизации с container registry.

ModuleUpdatePolicy

Short names: `mup`

Scope: Cluster

- `v1alpha2`
- `v1alpha1`

Определяет режим и настройки обновления модуля.

Пример:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
      - days:
        - Mon
        - Wed
      from: '13:30'
      to: '14:00'
```

- **spec**
объект

Обязательный параметр

- **spec.releaseChannel**
строка

Желаемый канал обновления модуля.

Порядок возрастания стабильности канала обновлений (от **наименее** стабильного к **наиболее** стабильному): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

По умолчанию: `Stable`

Допустимые значения: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `Lts`

- **spec.update**
объект

Обязательный параметр

Режим и настройки обновления модуля.

- **spec.update.mode**
строка

Режим обновления версий модуля.

- `AutoPatch` — автоматический режим обновления для патч-версий.

Для обновления минорной версии (например, с `v1.15.*` на `v1.16.*`) необходимо подтверждение.

Обновление патч-версии (например, с `v1.16.1` на `v1.16.2`) применяется с учетом окон обновлений, если они заданы.

- `Auto` (автоматический) — все обновления применяются автоматически.

Обновления минорной версии (например, с `v1.15.*` на `v1.16.*`) и патч-версии (например, с

v1.16.1 на v1.16.2) применяются с учетом заданных окон обновлений, либо, если окна обновлений не заданы, по мере появления обновлений на соответствующем канале обновлений.

- **Manual** (ручной) — для обновления и минорной, и патч-версии необходимо подтверждение.

Для подтверждения обновления установите аннотацию `modules.deckhouse.io/approved="true"` в соответствующем ресурсе `ModuleRelease`.

По умолчанию: `Auto`

Допустимые значения: `Auto`, `Manual`, `AutoPatch`

■ **spec.update.windows**

массив объектов

Расписание окон обновления модуля.

■ **spec.update.windows.days**

массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```

```
days: Wed
```

■ **Элемент массива**

строка

День недели.

Допустимые значения: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

Пример:

```
Mon
```

■ **spec.update.windows.from**

строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Должно быть меньше времени окончания окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

■ **spec.update.windows.to**

строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Должно быть больше времени начала окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Определяет режим и настройки обновления модуля.

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  moduleReleaseSelector:
    labelSelector:
      matchLabels:
        source: example
        module: module-1
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
      - days:
          - Mon
          - Wed
        from: '13:30'
        to: '14:00'

```

- **spec**

объект

Обязательный параметр

- **spec.moduleReleaseSelector**

объект

Обязательный параметр

Определяет модули к которым применяются настройки обновления.

- **spec.moduleReleaseSelector.labelSelector**

объект

Обязательный параметр

Определяет фильтр на основе лейблов, указанных у целевых модулей

Если указаны одновременно `matchExpressions` и `matchLabels`, учитываются оба фильтра (AND).

- **spec.moduleReleaseSelector.labelSelector.matchExpressions**

массив объектов

Список фильтров на основе выражений.

- **spec.moduleReleaseSelector.labelSelector.matchExpressions.key**

строка

Обязательный параметр

Имя лейбла.

- **spec.moduleReleaseSelector.labelSelector.matchExpressions.operator**

строка

Обязательный параметр

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.moduleReleaseSelector.labelSelector.matchExpressions.values**

массив строк

Значение лейбла.

- **spec.moduleReleaseSelector.labelSelector.matchLabels**

объект

Фильтр на основе совпадения лейблов.

Пример:

```

matchLabels:
  source: deckhouse
  module: deckhouse-admin

```

- **spec.releaseChannel**

строка

Желаемый канал обновления модуля. Порядок возрастания стабильности канала обновлений (от **наименее** стабильного к **наиболее** стабильному): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

По умолчанию: `Stable`

Допустимые значения: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`

◦ **spec.update**
объект

Обязательный параметр

Режим и настройки обновления модуля.

■ **spec.update.mode**
строка

Режим обновления версий модуля.

- `AutoPatch` — автоматический режим обновления для патч-версий.

Для обновления минорной версии (например, с `v1.15.*` на `v1.16.*`) необходимо подтверждение.

Обновление патч-версии (например, с `v1.16.1` на `v1.16.2`) применяется с учетом окон обновлений, если они заданы.

- `Auto` (автоматический) — все обновления применяются автоматически.

Обновления минорной версии (например, с `v1.15.*` на `v1.16.*`) и патч-версии (например, с `v1.16.1` на `v1.16.2`) применяются с учетом заданных окон обновлений, либо, если окна обновлений не заданы, по мере появления обновлений на соответствующем канале обновлений.

- `Manual` (ручной) — для обновления и минорной, и патч-версии необходимо подтверждение.

Для подтверждения обновления установите аннотацию `modules.deckhouse.io/approved="true"` в соответствующем ресурсе `ModuleRelease`.

- `Ignore` — обновления отключены.

По умолчанию: `AutoPatch`

Допустимые значения: `Auto`, `Manual`, `Ignore`, `AutoPatch`

■ **spec.update.windows**
массив объектов

Расписание окон обновления модуля.

■ **spec.update.windows.days**
массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```

```
days: Wed
```

■ **Элемент массива**
строка

День недели.

Допустимые значения: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

Пример:

```
Mon
```

■ **spec.update.windows.from**
строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Должно быть меньше времени окончания окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

■ **spec.update.windows.to**
строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Должно быть больше времени начала окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

PackageRepository

Scope: Cluster

Version: v1alpha1

PackageRepository является источником пакетов для Deckhouse Kubernetes Platform.

- **apiVersion**

строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации Kubernetes.

- **kind**

строка

Значение представляет REST-ресурс, который описывает данный объект. Серверы могут определить его по эндпоинту, к которому клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации Kubernetes.

- **metadata**

объект

- **spec**

объект

Обязательный параметр

Определяет конфигурацию репозитория пакетов.

- **spec.registry**

объект

Обязательный параметр

Конфигурация для доступа к хранилищу образов контейнеров с пакетами.

- **spec.registry.ca**

строка

Корневой сертификат (в формате PEM), которым можно проверить сертификат хранилища образов контейнеров при работе по HTTPS (если хранилище образов контейнеров использует самоподписанные SSL-сертификаты).

- **spec.registry.dockerCfg**

строка

Строка с токеном доступа к хранилищу образов контейнеров в Base64 (формат `~/docker/config.json`). В случае анонимного доступа к хранилищу образов контейнеров не заполняйте это поле.

- **spec.registry.login**

строка

Имя пользователя для аутентификации в хранилище образов контейнеров.

- **spec.registry.password**

строка

Пароль для аутентификации в хранилище образов контейнеров.

- **spec.registry.repo**

строка

Обязательный параметр

Адрес репозитория пакетов в хранилище образов контейнеров.

- **spec.registry.scheme**

строка

Протокол для доступа к репозиторию (например, `https`).

- **spec.scanInterval**

строка

Интервал сканирования хранилища образов контейнеров.

Определяет частоту проверки хранилища образов контейнеров на наличие новых пакетов.

По умолчанию: 6h

Шаблон: `^(\d+h)?(\d+m)?(\d+s)?$`

Примеры:

```
scanInterval: 5m
```

```
scanInterval: 1h
```

```
scanInterval: 6h30m
```

- **status**
объект

Статус репозитория пакетов.

- **status.conditions**
массив объектов

Условия, отражающие последние наблюдения за состоянием репозитория.

Условие, описывающее один аспект текущего состояния репозитория.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

Время последнего перехода условия из одного статуса в другой.

- **status.conditions.message**
строка

Обязательный параметр

Сообщение с деталями перехода. Может быть пустым.

Максимальная длина: 32768

- **status.conditions.observedGeneration**
целочисленный

Значение `.metadata.generation`, на котором было установлено условие. Позволяет определить актуальность условия для текущего состояния ресурса.

Допустимые значения: `0 <= X`

- **status.conditions.reason**
строка

Обязательный параметр

Программный идентификатор причины последнего перехода условия.

Шаблон: `^[A-Za-z]([A-Za-z0-9_]*[A-Za-z0-9_])?$`

Длина: 1..1024

- **status.conditions.status**
строка

Обязательный параметр

Статус условия. Может быть `True`, `False` или `Unknown`.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**
строка

Обязательный параметр

Тип условия репозитория в CamelCase или в формате `foo.example.com/CamelCase`.

Шаблон: `^([a-z0-9]([a-z0-9]*[a-z0-9])?(\.[a-z0-9]([a-z0-9]*[a-z0-9])?)*\/)?(((A-Za-z0-9)[:-A-Za-z0-9_]*)?[A-Za-z0-9])$`

Максимальная длина: 316

- **status.lastChangeTime**
строка

Время последнего сканирования, нашедшего хотя бы одну новую версию. Сканирования, не нашедшие нового, это значение не обновляют.

- **status.lastNewVersions**

целочисленный

Количество новых версий, найденных последним сканированием. Равно нулю, если последнее сканирование не нашло ничего нового.

- **status.lastScanTime**

строка

Время последнего сканирования с любым результатом.

- **status.message**

строка

Человеко-читаемое сообщение о статусе репозитория.

- **status.packages**

массив объектов

Список пакетов, доступных в этом репозитории.

- **status.packages.name**

строка

Обязательный параметр

Название пакета.

- **status.packages.type**

строка

Обязательный параметр

Тип пакета.

- **status.packagesCount**

целочисленный

Общее количество пакетов в этом репозитории.

- **status.partialScanAvailable**

булевый

Указывает, поддерживает ли хранилище образов контейнеров пагинацию при получении списка тегов.

- **status.phase**

строка

Текущая фаза репозитория.

PackageRepositoryOperation

Short names: pro

Scope: Cluster

Version: v1alpha1

PackageRepositoryOperation представляет операцию по сканированию/обновлению репозитория пакетов.

- **apiVersion**

строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации Kubernetes.

- **kind**

строка

Значение представляет REST-ресурс, который описывает данный объект. Серверы могут определить его по эндпоинту, к которому клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации Kubernetes.

- **metadata**

объект

- **spec**

объект

Обязательный параметр

Определяет параметры операции над репозиторием пакетов.

- **spec.packageRepositoryName**

строка

Обязательный параметр

Название репозитория пакетов для выполнения операции.

- **spec.type**

строка

Обязательный параметр

Тип выполняемой операции.

Допустимые значения: `Update`

- **spec.update**
объект

Конфигурация для операций обновления.

- **spec.update.fullScan**
булевый

Выполнять ли полное сканирование репозитория.

- **spec.update.timeout**
строка

Таймаут для операции.

- **status**
объект

Статус операции над репозиторием пакетов.

- **status.completionTime**
строка

Время завершения операции.

- **status.conditions**
массив объектов

Условия, отражающие последние наблюдения за состоянием операции. Фаза операции определяется по условию `Completed`: пока оно имеет статус `False`, операция находится в одной из промежуточных фаз (`Discover`, `Processing`); при статусе `True` операция завершена с причиной `Succeeded` или `Failed`.

Условие, описывающее один аспект текущего состояния операции.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

Время последнего перехода условия из одного статуса в другой.

- **status.conditions.message**
строка

Обязательный параметр

Сообщение с деталями перехода.

Максимальная длина: `32768`

- **status.conditions.observedGeneration**
целочисленный

Значение `.metadata.generation`, на котором было установлено условие. Позволяет определить актуальность условия для текущего состояния ресурса.

Допустимые значения: `0 <= X`

- **status.conditions.reason**
строка

Обязательный параметр

Программный идентификатор причины последнего перехода условия. Для условия `Completed` используются значения: `Discover`, `Processing`, `Succeeded`, `Failed`.

Шаблон: `^[A-Za-z]([A-Za-z0-9_]*[A-Za-z0-9_])?$`

Длина: `1..1024`

- **status.conditions.status**
строка

Обязательный параметр

Статус условия. Может быть `True`, `False` или `Unknown`.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**
строка

Обязательный параметр

Тип условия. Для данного ресурса используется условие `Completed`, которое отражает, достигнута ли терминальная фаза операции.

Шаблон: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*\/)?(((A-Za-z0-9)[:-A-Za-z0-9_\.]*)?[A-Za-z0-9])$`

Максимальная длина: 316

- **status.packages**
объект

Информация о пакетах, обработанных во время операции.

- **status.packages.discovered**
массив объектов

Список пакетов, обнаруженных во время операции.

- **status.packages.discovered.name**
строка

Обязательный параметр

Название обнаруженного пакета.

- **status.packages.failed**
массив объектов

Список пакетов, обработка которых не удалась.

- **status.packages.failed.errors**
массив объектов

Обязательный параметр

Список ошибок, возникших при обработке этого пакета.

- **status.packages.failed.errors.message**
строка

Обязательный параметр

Сообщение об ошибке.

- **status.packages.failed.errors.version**
строка

Обязательный параметр

Версия пакета, при обработке которой возникла ошибка.

- **status.packages.failed.name**
строка

Обязательный параметр

Название пакета, обработка которого не удалась.

- **status.packages.newVersionsOverall**
целочисленный

Общее количество новых версий по всем пакетам в рамках операции.

Версия считается новой, если её `ApplicationPackageVersion` или `ModulePackageVersion` отсутствовал в кластере, либо (только для `ApplicationPackageVersion`) имел пометку «нет в реестре» и его образ был найден в реестре в рамках этой операции.

- **status.packages.processed**
массив объектов

Список успешно обработанных пакетов.

- **status.packages.processed.foundVersions**
целочисленный

Количество версий, найденных в рамках данной операции.

- **status.packages.processed.name**
строка

Обязательный параметр

Название обработанного пакета.

- **status.packages.processed.newVersions**
целочисленный

Количество новых версий в рамках данной операции.

Версия считается новой, если её `ApplicationPackageVersion` или `ModulePackageVersion` отсутствовал в кластере, либо (только для `ApplicationPackageVersion`) имел пометку «нет в реестре» и его образ был найден в реестре в рамках этой операции.

- **status.packages.processed.type**

строка

Тип пакета.

- **status.packages.processedOverall**

целочисленный

Общее количество обработанных пакетов.

- **status.packages.total**

целочисленный

Общее количество найденных пакетов.

- **status.startTime**

строка

Время начала операции.

SSHConfig

Version: dhctl.deckhouse.io/v1

Конфигурация SSH для dhctl.

Пример:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHConfig
sshUser: user
sshPort: 22
sshExtraArgs: "-vvv"
sshAgentPrivateKeys:
- key: "<ssh-private-key>"
```

- **apiVersion**

строка

Версия Deckhouse API.

Допустимые значения: `dhctl.deckhouse.io/v1`

- **kind**

строка

Допустимые значения: `SSHConfig`

- **legacyMode**

булевый

Использовать легаси режим SSH (clissh).

- **modernMode**

булевый

Использовать актуальный режим SSH (gossh).

- **sshAgentPrivateKeys**

массив объектов

- **sshAgentPrivateKeys.key**

строка

Обязательный параметр

Приватный SSH-ключ.

- **sshAgentPrivateKeys.passphrase**

строка

Пароль SSH-ключа.

- **sshBastionHost**

строка

Хост SSH-бастиона.

- **sshBastionPassword**

строка

A password for the bastion user.

- **sshBastionPort**

целочисленный

Порт SSH-бастиона.

- **sshBastionUser**

строка

Имя пользователя бастiona.

- **sshExtraArgs**

строка

Дополнительные параметры соединения SSH.

- **sshPort**

целочисленный

Порт SSH.

- **sshUser**

строка

Имя пользователя SSH.

- **sudoPassword**

строка

Пароль sudo пользователя.

SSHHost

Version: dhctl.deckhouse.io/v1

Конфигурация SSH-хоста.

Пример:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHHost
host: 172.16.0.0
```

- **apiVersion**

строка

Обязательный параметр

Версия Deckhouse API.

Допустимые значения: `dhctl.deckhouse.io/v1`

- **host**

строка

Обязательный параметр

SSH-хост.

- **kind**

строка

Обязательный параметр

Допустимые значения: `SSHHost`

StaticClusterConfiguration

Version: deckhouse.io/v1

Параметры статического (bare metal) кластера.

Чтобы изменить содержимое ресурса `StaticClusterConfiguration` в работающем кластере, выполните следующую команду:

```
d8 system edit static-cluster-configuration
```

Пример:

```
apiVersion: deckhouse.io/v1
kind: StaticClusterConfiguration
internalNetworkCIDRs:
- 10.244.0.0/16
- 10.50.0.0/16
```

- **apiVersion**

строка

Обязательный параметр

Используемая версия API Deckhouse.

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **internalNetworkCIDRs**

массив строк

Список внутренних сетей узлов кластера.

Внутренние сети используются для связи компонентов Kubernetes (`kube-apiserver`, `kubelet` и т. д.) между собой.

Параметр обязателен к заполнению, если:

- Узлы кластера имеют более одного сетевого интерфейса
- Развертывание кластера происходит внутри Deckhouse Virtualization Platform



При смене значения, в список подсетей должны подпадать существующие адреса узлов кластера.

Пример:

```
internalNetworkCIDRs:
- 192.168.42.0/24
- 172.16.16.0/24
```

- **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|1-2[0-9]|0-9))$`

- **kind**

строка

Обязательный параметр

Допустимые значения: `StaticClusterConfiguration`

Каналы обновлений



Информацию о том, какие версии Deckhouse находятся в настоящий момент на каких каналах обновлений, а также о планируемой дате смены версии на канале обновлений смотрите на сайте releases.deckhouse.ru.

К кластерам как элементам инфраструктуры обычно предъявляются различные требования.

Например, production-кластер, в отличие от кластера разработки, более требователен к надежности: в нем нежелательно часто обновлять или изменять какие-либо компоненты без особой необходимости, при этом сами компоненты должны быть тщательно протестированы.

Deckhouse использует **пять каналов обновлений**. Мягко переключаться между ними можно с помощью модуля `deckhouse`: достаточно указать желаемый канал обновлений в конфигурации модуля.

- Rock Solid
- Stable
- Early Access
- Beta
- Alpha

- Наиболее стабильный канал обновлений. Подойдет для кластеров, которым необходимо обеспечить повышенный уровень стабильности. Обновления функционала до этого канала доходят не ранее чем через месяц после их появления в релизе.
- Стабильный канал обновлений для кластеров, в которых закончена активная работа и преимущественно осуществляется эксплуатация. Обновления функционала до этого канала обновлений доходят не ранее чем через две недели после их появления в релизе.
- **Рекомендуемый** канал обновлений, если вы не уверены в выборе. Подойдет для кластеров, в которых идет активная работа (запускаются, дорабатываются новые приложения и т. п.). Обновления функционала до этого канала обновлений доходят не ранее чем через одну неделю после их появления в релизе.
- Ориентирован на кластеры разработки, как и канал обновлений Alpha. Получает версии, предварительно опробованные на канале обновлений Alpha.
- Наименее стабильный канал обновлений с наиболее частым появлением новых версий. Ориентирован на кластеры разработки с небольшим количеством разработчиков.

Дополнительные ресурсы

- releases.deckhouse.ru
- Документация модуля `deckhouse`

Поддерживаемые версии Kubernetes и ОС

Linux

В настоящий момент в качестве ОС для узлов поддерживаются следующие дистрибутивы Linux:

Дистрибутив Linux	Поддерживаемые версии	Поддерживается в редакциях			Примечания
		CE	CSE Lite, CSE Pro	BE, SE, SE+, EE	
МОС ОС	15.4, 15.5	~ Работоспособность в Community Edition не гарантируется.	✓	✓	
ALT Linux	р10, 10.0, 10.1, 10.2, 11	✓	✓	✓	
	8 СП (релиз 10)	~ Работоспособность в Community Edition не гарантируется.	✓	✓	
Astra Linux Special Edition	1.7, 1.8	✓	✓	✓	
CentOS	7, 8, 9	✓	✗	✓	
Debian	10, 11, 12, 13	✓	✗	✓	
openSUSE	15.4, 15.5, 15.6	✓	✗	✓	
РЕД ОС	7.3, 8.0	✓	✓	✓	
Rocky Linux	8, 9	✓	✗	✓	
POCA Сервер	7.9, 12.4, 12.5, 12.6	✓	✗	✓	
SberLinux	9.8	✓	✗	✓	
Ubuntu	18.04, 20.04, 22.04, 24.04	✓	✗	✓	

Kubernetes

В настоящий момент поддерживаются следующие версии Kubernetes:

Версия	Статус
1.31	Версия не рекомендуется для использования. Запланирована для удаления в будущих релизах.
1.32	Поддерживаемая версия, готовая для использования в production-окружениях.
1.33	Версия по умолчанию. Поддерживаемая версия, готовая для использования в production-окружениях.
1.34	Поддерживаемая версия, готовая для использования в production-окружениях.
1.35	Версия с ранним доступом, которая не подходит для использования в production-окружениях.

Container registry



Deckhouse Kubernetes Platform работает с любыми container registry, поддерживающими схемы авторизации Basic или Bearer token и OCI формат образов.

Container registry, с которыми Deckhouse гарантированно работает
Sonatype Nexus Repository
Harbor
JFrog Artifactory
CNCF Distribution Registry (Docker Registry)
Project Quay*

Сравнение редакций Deckhouse Kubernetes Platform

Deckhouse Kubernetes Platform поставляется в нескольких редакциях. Они отличаются набором возможностей и уровнем доступной поддержки. Краткое сравнение ключевых возможностей и особенностей можно найти на сайте Deckhouse Kubernetes Platform.

В таблице ниже перечислены модули Deckhouse Kubernetes Platform, доступные для использования в разных редакциях платформы:

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
adaptive-resource-management	✓	✓	✓	✓	✓	✗	✗
admission-policy-engine	~	~	~	~	✓	✓	✓
	Ограничение на возможность использования политик пров ерки подп иси обра зов конт ейне ров и возм ожно сть уста новк и запр ета на запус к конт ейне ров с уязв имостями	Ограничение на возможность использования политик пров ерки подп иси обра зов конт ейне ров и возм ожно сть уста новк и запр ета на запус к конт ейне ров с уязв имостями	Ограничение на возможность использования политик пров ерки подп иси обра зов конт ейне ров и возм ожно сть уста новк и запр ета на запус к конт ейне ров с уязв имостями	Ограничение на возможность использования политик пров ерки подп иси обра зов конт ейне ров и возм ожно сть уста новк и запр ета на запус к конт ейне ров с уязв имостями			
ai-models	✗	✗	✗	✗	✓	✗	✗
alb	✓	✓	✓	✓	✓	✗	✗
ansible	✓	✗	✗	✓	✓	✗	✗
cert-manager	✓	✓	✓	✓	✓	✓	✓
chrony	✓	✓	✓	✓	✓	✓	✓
cilium-hubble	✓	✓	✓	✓	✓	✓	✓
cloud-provider-aws	✓	✓	✓	✓	✓	✗	✗
cloud-provider-azure	✓	✓	✓	✓	✓	✗	✗
cloud-provider-dyp	✓	✓	✓	✓	✓	✓	✓
cloud-provider-dynamix	✗	✗	✗	✗	✓	✗	✗
cloud-provider-gcp	✓	✓	✓	✓	✓	✗	✗
cloud-provider-huaweicloud	✗	✗	✗	✗	✓	✗	✗
cloud-provider-openstack	✗	✗	✗	✗	✓	✗	✗
cloud-provider-vcd	✗	✗	✗	✗	✓	✗	✗
cloud-provider-vsphere	✗	✗	✗	✓	✓	✗	✗

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
cloud-provider-yandex	✓	✓	✓	✓	✓	✗	✗
cloud-provider-zvirt	✗	✗	✗	✓	✓	✗	✗
cni-cilium	~ Ограничение на возможность настройки и отказоустойчивых выходов узлов (Egress Gateway)	~ Ограничение на возможность настройки и отказоустойчивых выходов узлов (Egress Gateway)	~ Ограничение на возможность настройки и отказоустойчивых выходов узлов (Egress Gateway)	✓	✓	✓	✓
cni-flannel	✓	✓	✓	✓	✓	✗	✗
cni-simple-bridge	✓	✓	✓	✓	✓	✗	✗
code	✗	✗	~ Требуется отдельная лицензия	~ Требуется отдельная лицензия	~ Требуется отдельная лицензия	✗	✗
commander	~ Требуется отдельная лицензия	~ Требуется отдельная лицензия	~ Требуется отдельная лицензия	✓	✓	✓	✓
commander-agent	✓	✓	✓	✓	✓	✓	✓
console	✓	✓	✓	✓	✓	✓	✓
control-plane-manager	✓	✓	✓	✓	✓	✓	✓
csi-ceph	✓	✗	✓	✓	✓	✓	✓
csi-hpe	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
csi-huawei	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
csi-netapp	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
csi-nfs	✓	✗	✓	✓	✓	✓	✓
csi-s3	✗	✗	✗	✗	✓	✗	✗
csi-scsi-generic	✗	✗	✗	✗	✓	✓	✓
csi-vsphere	✗	✗	✗	✓	✓	✗	✗
csi-yadro-tatlin-unified	✗	✗	✗	✗	✓	✓	✓
dashboard	✓	✓	✗	✗	✓	✗	✗
deckhouse	✓	✓	✓	✓	✓	✓	✓
deckhouse-tools	✓	✓	✓	✓	✓	✓	✓
descheduler	✓	✓	✓	✓	✓	✓	✓
development-platform	✗	✗	✗	✗	✓	✗	✗
documentation	✓	✓	✓	✓	✓	✓	✓

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
extended-monitoring	✓	✓	✓	✓	✓	✓	✓
gpu	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
ingress-nginx	✓	✓	✓	✓	✓	✓	✓
istio	~ Ограничение на использование федерации и мультикластерной конфигурации	~ Ограничение на использование федерации и мультикластерной конфигурации	~ Ограничение на использование федерации и мультикластерной конфигурации	~ Ограничение на использование федерации и мультикластерной конфигурации	✓	~ Ограничение на использование федерации и мультикластерной конфигурации	✓
keepalived	✗	✗	✗	✗	✓	✗	✗
kube-dns	✓	✓	✓	✓	✓	✓	✓
kube-proxy	✓	✓	✓	✓	✓	✗	✗
local-path-provisioner	✓	✓	✓	✓	✓	✓	✓
log-shipper	✓	✓	✓	✓	✓	✓	✓
loki	✓	✓	✓	✓	✓	✓	✓
managed-cassandra	✗	✗	✗	✗	✓	✗	✗
managed-clickhouse	✗	✗	✗	✗	✓	✗	✗
managed-hive-metastore	✗	✗	✗	✗	✓	✗	✗
managed-kafka	✗	✗	✗	✗	✓	✗	✗
managed-memcached	✗	✗	✓	✓	✓	✗	✗
managed-opensearch	✗	✗	✗	✗	✓	✗	✗
managed-postgres	✗	✗	✓	✓	✓	✗	✗
managed-starrocks	✗	✗	✓	✓	✓	✗	✗
managed-trino	✗	✗	✗	✗	✓	✗	✗
managed-valkey	✗	✗	✓	✓	✓	✗	✗
metallb	✗	✗	~ Ограничение на использование BGP-балансировщика для сервисов	~ Ограничение на использование BGP-балансировщика для сервисов	✓	~ Ограничение на использование BGP-балансировщика для сервисов	✓
monitoring-custom	✓	✓	✓	✓	✓	✓	✓
monitoring-deckhouse	✓	✓	✓	✓	✓	✓	✓
monitoring-kubernetes	✓	✓	✓	✓	✓	✓	✓
monitoring-kubernetes-control-plane	✓	✓	✓	✓	✓	✓	✓
monitoring-ping	✓	✓	✓	✓	✓	✓	✓
multitenancy-manager	✓	✓	✓	✓	✓	✓	✓
namespace-configurator	✓	✓	✓	✓	✓	✓	✓
network-gateway	✗	✗	✗	✗	✓	✗	✗
network-policy-engine	✓	✓	✓	✓	✓	✗	✗
neuvevector	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
node-local-dns	✗	✓	✓	✓	✓	✓	✓
node-manager	✓	✓	✓	✓	✓	✓	✓

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
observability	✓	✓	✓	✓	✓	✗	✓
observability-platform	✗	✗	✗	✗	~ Требуется отдельная лицензия	✗	✗
okmeter	✓	✓	✓	✓	✓	✗	✗
openvpn	✓	✓	✓	✓	✓	✗	✗
operator-argo	✗	✗	✗	✗	✓	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника	~ Не входит в состав редакции. Может быть использован как модуль, подключаемый из внешнего источника
operator-helm	✓	✓	✓	✓	✓	✗	✗
operator-prometheus	✓	✓	✓	✓	✓	✓	✓
operator-trivy	✗	✗	✗	✗	✓	✓	✓
payload-registry	✗	✗	✗	✗	✓	✗	✗
pod-reloader	✓	✓	✓	✓	✓	✗	✗
prometheus	✓	✓	✓	✓	✓	✓	✓
prometheus-metrics-adapter	✓	✓	✓	✓	✓	✓	✓
prometheus-pushgateway	✓	✓	✓	✓	✓	✗	✗
prompp	✓	✓	✓	✓	✓	✓	✓
registry	✓	✓	✓	✓	✓	✓	✓
registry-packages-proxy	✓	✓	✓	✓	✓	✓	✓
runtime-audit-engine	✗	✗	✗	✗	✓	✓	✓
sdn	~ Есть ограниче ния функцион альности ости	~ Есть ограниче ния функцион альности ости	~ Есть ограниче ния функцион альности ости	~ Есть ограниче ния функцион альности ости	✓	✓	✓
sds-elastic	✗	✗	✗	✗	✓	✗	✗
sds-local-volume	~ Ограниче ние на возмож ность создания снимков и затирани е томов	✗	✓	✓	✓	✓	✓
sds-node-configurator	✓	✗	✓	✓	✓	✓	✓
sds-object	✗	✗	✗	✗	✓	✗	✗
sds-replicated-volume	~ Ограниче ние на возмож ность создания снимков	✗	✓	✓	✓	✗	✗
secret-copier	✓	✓	✓	✓	✓	✓	✓
secrets-store-integration	✓	✓	✓	✓	✓	✓	✓
security-events-manager	✗	✗	✗	✗	✓	✗	✗
service-with-healthchecks	✗	✗	✗	✗	✓	✓	✓
snapshot-controller	✓	✓	✓	✓	✓	✓	✓
state-snapshotter	✓	✓	✓	✓	✓	✗	✗
static-routing-manager	✗	✗	✗	✗	✓	✗	✗
storage-foundation	✓	✓	✓	✓	✓	✗	✗

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
storage-volume-data-manager	X	X	X	X	✓	✓	✓
stronghold	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.	~ Некоторые функции доступны в зависимости от выбранной лицензии Deckhouse Stronghold. Подробнее можно ознакомиться в разделе «Редакции» документации Deckhouse Stronghold.
terraform-manager	✓	✓	✓	✓	✓	✓	✓
upmeter	✓	✓	✓	✓	✓	✓	✓
user-authn	✓	✓	✓	✓	✓	✓	✓
user-authz	~ Ограничение на использование функций по пространствам имен в ролевой модели доступа	✓	✓	✓	✓	✓	✓
vertical-pod-autoscaler	✓	✓	✓	✓	✓	✓	✓

	CE	BE	SE	SE+	EE	CSE Lite (1.73)	CSE Pro (1.73)
virtualization	~ Есть огра ниче ния функ цион альн ости	X	X	✓	✓	X	✓

Структура хранилища образов контейнеров

На странице представлено общее описание хранилищ образов контейнеров, из которых производится установка Deckhouse Kubernetes Platform (DKP).

Хранилище образов сертифицированной версии

Хранилище образов Deckhouse Kubernetes Platform Certified Security Edition (DKP CSE, сертифицированная ФСТЭК России редакция DKP) расположено по адресу `registry-cse.deckhouse.ru/deckhouse/cse`. Оно одержит все модули и компоненты, включенные в комплект поставки.

Ключевые особенности:

- проходит сертификационные испытания;
- входит в комплект поставки DKP CSE;
- реализует все заявленные функциональные возможности и функции безопасности.

К образам предоставляется техническая поддержка, включающая консультации по вопросам эксплуатации.

Особенности сборки и обновления образов:

- сборка производится из исходных кодов открытого ПО, входящего в состав DKP:
 - из upstream («как есть»);
 - с доработками к применению в DKP CSE;
 - с глубокой модернизацией (fork).
- в состав входят собственные разработки компании;
- устранение уязвимостей, разработка компенсирующих мер и обновление пакетов осуществляется в сроки, установленные в Формуляре на DKP CSE;
- предоставляется доступ к обновлениям:
 - с новой функциональностью;
 - с исправлениями ошибок и уязвимостей;
 - с решением инцидентов безопасности.

Хранилище образов коммерческих версий

Хранилище содержит модули и компоненты, доступные в коммерческих редакциях DKP:

- EE (адрес `registry.deckhouse.ru/deckhouse/ee`);
- SE (адрес `registry.deckhouse.ru/deckhouse/se`);
- SE+ (адрес `registry.deckhouse.io/deckhouse/se-plus`);
- BE (адрес `registry.deckhouse.ru/deckhouse/be`).

Ключевые особенности:

- реализует функциональные возможности согласно редакции;
- для модулей, входящих в состав DKP CSE, проводится тестирование безопасности в рамках цикла безопасной разработки ПО;
- поставляется в электронном виде.

К образам предоставляется техническая поддержка, включающая консультации по вопросам эксплуатации.

Особенности сборки и обновления образов:

- сборка производится из исходных кодов открытого ПО, входящего в состав DKP:
 - из upstream («как есть»);
 - с доработками к применению в DKP CSE;
 - с глубокой модернизацией (fork).
- в состав входят собственные разработки компании;
- устранение уязвимостей и обновление пакетов выполняется только по мере обновления исходных пакетов открытого ПО;
- предоставляется доступ к обновлениям:
 - с новой функциональностью;
 - с исправлениями ошибок и уязвимостей;
 - с решением инцидентов безопасности.

Сетевое взаимодействие компонентов платформы

Если на площадке, где работает Deckhouse Kubernetes Platform (DKP), есть требования для ограничения сетевого взаимодействия между серверами на уровне инфраструктуры, то необходимо соблюсти следующие условия:

- Включен режим туннелирования трафика между подами (настройки для CNI Cilium, настройки для CNI Flannel).
- Разрешена передача трафика между podSubnetCIDR, инкапсулированного внутри VXLAN (если выполняется

- инспектирование и фильтрация трафика внутри VXLAN-туннеля).
- В случае необходимости интеграции с внешними системами (например, LDAP, SMTP или прочие внешние API), с ними разрешено сетевое взаимодействие.
- Локальное сетевое взаимодействие полностью разрешено в рамках каждого отдельно взятого узла кластера.
- Разрешено взаимодействие между узлами по портам, приведенным в таблицах на текущей странице. Обратите внимание, что большинство портов входит в диапазон 4200-4299. При добавлении новых компонентов платформы им будут назначаться порты из этого диапазона (при наличии возможности).

Как проверить текущий порт VXLAN...

```
d8 k -n d8-cni-cilium get cm cilium-config -o yaml | grep tunnel
```

Пример вывода команды:

```
routing-mode: tunnel
tunnel-port: "4298"
tunnel-protocol: vxlan
```

i Изменения, связанные с добавлением, удалением или переопределением портов в таблицах, перечислены в подразделе «Сеть» соответствующей версии DKP на странице «История изменений».

Требования к сетевым задержкам между master-узлами

i В DKP поддерживается режим HA с использованием arbiter-узлов. Приведенные ниже требования к сетевым задержкам также применимы и к arbiter-узлам.

Etcd подтверждает запись только после того, как изменение реплицировано на большинство master-узлов. Поэтому круговая задержка сети (RTT, round-trip time) между master-узлами напрямую определяет скорость работы кластера. Одна операция в кластере обычно состоит из нескольких последовательных записей, поэтому с ростом задержки заметно замедляются применение манифестов, работа операторов и отклик Kubernetes API.

Сумма RTT и джиттера между master-узлами не должна превышать 100 мс.

 Указана круговая задержка (RTT), а не задержка в одну сторону: например команда `ping` возвращает RTT. Джиттер — разброс задержки — добавляется к RTT. Например, при RTT 90 мс и джиттере 20 мс сумма составляет 110 мс, и требование не выполняется, хотя само значение RTT укладывается в 100 мс.

При планировании кластера рекомендуется добиваться задержки не более 50 мс между будущими master-узлами, то есть двукратного запаса к требованию. Запас необходим по следующим причинам:

- Измерения до установки выполняются на незагруженной сети и могут занижить реальные значения.
- Нужно учитывать всплески и разброс задержки, а также возможные операции с инфраструктурой — например, миграцию виртуальной машины в зону с более высокой задержкой.

Откуда взято значение 100 мс...

DKP использует стандартные значения etcd:

- `ETCD_HEARTBEAT_INTERVAL` — 100 мс, частота отправки heartbeat-сообщений от лидера остальным членам кластера.
- `ETCD_ELECTION_TIMEOUT` — 1000 мс, время ожидания heartbeat-сообщений, после которого член кластера начинает перевыборы лидера.

Документация etcd требует, чтобы таймаут перевыборов был не менее 10x RTT. Отсюда максимально допустимое значение:

```
1000 мс / 10 = 100 мс
```

Возможные проблемы

Главное следствие повышенного RTT — пропорциональное замедление кластера. Задержка сети переходит во время записи один к одному: при RTT 300 мс каждая запись в etcd займёт не менее 300 мс. Порога здесь нет, деградация идёт непрерывно с ростом задержки.

Вторая проблема — перевыборы лидера etcd. Вызываются они не величиной задержки, а её полным пропаданием: член кластера начинает перевыборы, если не получает сообщений от лидера в течение времени, заданного `ETCD_ELECTION_TIMEOUT` (с учётом встроенного в etcd механизма рандомизации — от 1000 до 2000 мс). При интервале heartbeat 100 мс это соответствует 10–20 не полученным подряд heartbeat-сообщениям. Типичные причины:

- обрыв связи или потеря пакетов на протяжении 1–2 секунд;
- задержки записи на диск на узле-лидере, из-за которых etcd не успевает отправлять heartbeat-сообщения;
- перезапуск пода etcd.

Отдельные всплески задержки к перевыборам не приводят: heartbeat-сообщения приходят, но с опозданием. Замедляются только записи, попавшие в момент всплеска.

Однако высокий RTT делает перевыборы затяжными, когда они всё же случаются. Голоса собираются за время одного RTT и должны уложиться в `ETCD_ELECTION_TIMEOUT`. Если RTT близок к этому таймауту, перевыборы не успевают завершиться и начинаются заново, а кластер остаётся без лидера. Десятикратный запас нужен, чтобы перевыборы прошли с первой попытки.

Во время перевыборов `etcd` не обслуживает запросы на запись, поэтому Kubernetes API возвращает ошибки. О частых перевыборах сигнализирует алерт `KubeEtcdHighNumberOfLeaderChanges` — более трёх перевыборов лидера за 10 минут.

Проверка текущих задержек

Задержку и её разброс можно измерить с помощью `ping`. Команда выполняется на одном из master-узлов, а при планировании кластера — на сервере, который предполагается сделать master-узлом. Для такой проверки достаточно ICMP, который уже входит в требования к трафику между узлами:

```
ping -c 100 -i 0.2 <IP-адрес другого master-узла>
```

Пример итоговой строки вывода:

```
rtt min/avg/max/mdev = 25.085/25.502/35.933/1.185 ms
```

Здесь `avg` — средний RTT, `mdev` — разброс задержки (джиттер), `max` — фактический максимум, то есть RTT с учётом всплесков. Складывать RTT и джиттер вручную не нужно: сравнивать с требованиями следует значение `max`. При планировании кластера рекомендуем укладываться в 50 мс, на работающем кластере — не превышать 100 мс.

Сетевые задержки на остальных узлах

Для работы control plane требований к сетевым задержкам на узлах, не являющихся master-узлами, нет: величина задержки на них не влияет на устойчивость кластера. Значение имеет только длительность потери связи с master-узлами — по умолчанию через 40 секунд узел переходит в состояние `Unreachable`, а через 5 минут его поды переносятся на другие узлы. Оба порога настраиваются параметрами `nodeMonitorGracePeriodSeconds` и `failedNodePodEvictionTimeoutSeconds`.

При этом задержки влияют на скорость работы всего, что запущено на узлах, — в том числе компонентов самой платформы. Для компонентов, обращающихся к Kubernetes API (операторы DKP, Prometheus, Ingress-контроллеры), задержка до master-узлов добавляется к каждому запросу. Задержка между узлами добавляется к каждому сетевому запросу между подами и к разрешению DNS-имён, а при использовании реплицируемого хранилища — ко времени записи на диск.

Таким образом допустимое значение определяется требованиями запускаемой нагрузки. Для кластеров, распределённых между удалёнными площадками, это означает, что платформа продолжит работать, но задержки приложений будут отражать географию размещения узлов.

Трафик между master-узлами

Порт	Протокол	Назначение
2379, 2380	TCP	Репликация etcd
4200	TCP	Вебхук-обработчик Cluster API
4201	TCP	Вебхук-обработчик для cloud-провайдера VMware Cloud Director
4223	TCP	Вебхук-обработчик контроллера Deckhouse

Трафик от master-узлов к узлам

Порт	Протокол	Назначение
22	TCP	SSH для первичной настройки статических узлов статичным провайдером
10250	TCP	kubelet

Порт	Протокол	Назначение
4221	TCP	apiserver bashible для доставки конфигурации на узлы
4227	TCP	Вебхук-обработчик компонента runtime-audit-engine

Трафик от узлов к master-узлам

Порт	Протокол	Назначение
4234	UDP	NTP для синхронизации времени между узлами
6443	TCP	kube-apiserver для контроллеров, работающих в сетевом пространстве имен узла
4203	TCP	Метрики компонента machine-controller-manager
4219	TCP	Прокси для пакетов registry registry-packages-proxy
4222	TCP	Метрики контроллера Deckhouse

Трафик между узлами

Порт	Протокол	Назначение
—	ICMP	ICMP для мониторинга связности между узлами
4202	TCP	Метрики агента sds-node-configurator
4204	TCP	Debug для контроллера Deckhouse
4205	TCP	Метрики модуля ebpf-exporter
4206	TCP	Метрики модуля node-exporter
4207	TCP	Метрики контроллера ingress-nginx для инлета HostWithFailover
4208	TCP	Метрики контроллера ingress-nginx для инлета HostWithFailover
4209	TCP	Метрики управляющего слоя Kubernetes
4210	TCP	Метрики kube-proxy
4211	TCP	Метрики Cluster API

Порт	Протокол	Назначение
4212	TCP	Метрики модуля <code>runtime-audit-engine</code>
4213	TCP	Метрики <code>kube-router</code>
4214	TCP	API агента модуля <code>sds-replicated-volume</code>
4215	TCP	Метрики агента <code>sds-replicated-volume</code>
4216	TCP	Метрики агента <code>storage-status</code>
4218	TCP/UDP	Синхронизация компонентов <code>speaker</code> модулей <code>metallb</code> через протокол <code>memberlist</code>
4220	TCP	Метрики компонентов <code>speaker</code> модулей <code>metallb</code>
4224	TCP	Метрики <code>node-local-dns</code>
4225	TCP/UDP	Синхронизация компонентов <code>speaker</code> модулей <code>metallb</code> через протокол <code>memberlist</code>
4226	TCP	Метрики компонентов <code>speaker</code> модулей <code>metallb</code>
4228	TCP	Healthcheck агента модуля <code>sds-node-configurator</code>
4229	TCP	Healthcheck CSI-контроллера модуля <code>csi-nfs</code>
4230	TCP	Healthcheck CSI-агентов модуля <code>csi-nfs</code>
4231	TCP	Healthcheck CSI-контроллера модуля <code>csi-hpe</code>
4232	TCP	Healthcheck CSI-агентов модуля <code>csi-hpe</code>
4235	TCP	Healthcheck CSI-контроллера модуля <code>csi-s3</code>
4236	TCP	Healthcheck CSI-агентов модуля <code>csi-s3</code>
4237	TCP	Healthcheck CSI-контроллера модуля <code>csi-scsi-generic</code>
4238	TCP	Healthcheck CSI-агентов модуля <code>csi-scsi-generic</code>

Порт	Протокол	Назначение
4239	TCP	Healthcheck агента модуля <code>storage-status</code>
4240	TCP	Порт для процедуры healthcheck соседних узлов в CNI Cilium
4241	TCP	Метрики агентов CNI Cilium
4242	TCP	Метрики оператора CNI Cilium
4244	TCP	API для модуля <code>cilium-hubble</code>
4245	TCP	Метрики <code>chrony-exporter</code>
4246	TCP	Healthcheck CSI-контроллера CephFS модуля <code>csi-ceph</code>
4247	TCP	Healthcheck CSI-контроллера RBD модуля <code>csi-ceph</code>
4248	TCP	Healthcheck CSI-контроллера модуля <code>csi-yadro-tatlin-unified</code>
4249	TCP	Healthcheck CSI-агентов модуля <code>csi-yadro-tatlin-unified</code>
4250	TCP	Healthcheck CSI-контроллера модуля <code>sds-local-volume</code>
4251	TCP	Healthcheck CSI-агентов модуля <code>sds-local-volume</code>
4252	TCP	Healthcheck CSI-агентов RBD модуля <code>csi-ceph</code>
4253	TCP	Healthcheck CSI-агентов CephFS модуля <code>csi-ceph</code>
4254	TCP	Healthcheck CSI-контроллера модуля <code>csi-netapp</code>
4255	TCP	Healthcheck CSI-агентов модуля <code>csi-netapp</code>
4256	TCP	Метрики CSI-контроллера модуля <code>csi-netapp</code>
4257	TCP	Порт API CSI-контроллера модуля <code>csi-netapp</code>
4258	TCP	Порт вебхука CSI-контроллера модуля <code>csi-huawei</code>
4259	TCP	Healthcheck CSI-агентов модуля <code>csi-huawei</code>

Порт	Протокол	Назначение
4260	TCP	Метрики CSI-контроллера модуля <code>csi-huawei</code>
4261	TCP	Healthcheck CSI-контроллера модуля <code>sds-replicated-volume</code>
4262	TCP	Healthcheck CSI-агентов модуля <code>sds-replicated-volume</code>
4263	TCP	Метрики модуля <code>service-with-healthchecks</code>
4269	TCP	Healthcheck CSI-агентов модуля <code>sds-replicated-volume</code>
4270	TCP	Метрики CSI-агентов модуля <code>sds-replicated-volume</code>
4271	TCP	Healthcheck CSI-контроллера модуля <code>sds-replicated-volume</code>
4272	TCP	Метрики CSI-контроллера модуля <code>sds-replicated-volume</code>
4135-4199	TCP	Туннели live migration модуля <code>virtualization</code> между узлами кластера
4280	TCP	Протокол USB over IP (<code>usb/ip</code>) модуля <code>virtualization</code> для проброса USB-устройств между узлами кластера
4286	TCP	Метрики Istio CNI
4287	UDP	Порт WireGuard для шифрования трафика в CNI Cilium
4288	TCP	Метрики <code>monitoring-ping</code>
4289	TCP	Метрики <code>monitoring-ping</code>
4295-4297	UDP	Используется модулем <code>cni-cilium</code> для VXLAN-инкапсуляции трафика между подами при множественной вложенной виртуализации — когда DKP с включенным модулем <code>virtualization</code> развернут внутри виртуальных машин, также созданных в DKP с включенным модулем <code>virtualization</code>
4298	UDP	Используется модулем <code>cni-cilium</code> для VXLAN-инкапсуляции трафика между подами, если кластер был развернут на DKP, начиная с версии 1.71 (для кластеров, развернутых на DKP до версии 1.71 , см. примечание для портов 4299/UDP, 8469/UDP и 8472/UDP)

Порт	Протокол	Назначение
4299	UDP	Для кластеров, развернутых на DKP версий 1.64-1.70. Используется модулем <code>cni-cilium</code> для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт, если не включается модуль <code>virtualization</code>. Обратите внимание, что в таких кластерах включение модуля <code>virtualization</code> на DKP до версии 1.70 меняет порт на 4298/UDP
7000-7999	TCP	Репликация DRBD для <code>sds-replicated-volume</code>
8469	UDP	Для кластеров, развернутых на DKP версии 1.63 и ниже с модулем <code>virtualization</code>, включенным до DKP версии 1.63. Используется модулем <code>cni-cilium</code> для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт
8472	UDP	Для кластеров, развернутых на DKP версии 1.63 и ниже. Используется модулем <code>cni-cilium</code> для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт, если не включается модуль <code>virtualization</code>. Обратите внимание, что в таких кластерах включение модуля <code>virtualization</code> на DKP до версии 1.70 меняет порт: • включение модуля <code>virtualization</code> на DKP версии 1.63 и ниже изменит его на 8469/UDP и не изменит при последующих обновлениях DKP • включение модуля <code>virtualization</code> на DKP, начиная с версии 1.64, изменит его на 4298/UDP и не изменит при последующих обновлениях DKP

Статические порты модуля `virtualization` на узлах кластера (зарезервирован диапазон 4100-4134/TCP)

Порт	Протокол	Назначение
4100	TCP	Healthz и метрики <code>virt-handler</code> модуля <code>virtualization</code>
4101	TCP	Сервер консоли <code>virt-handler</code> модуля <code>virtualization</code>

Порт	Протокол	Назначение
4105	TCP	Порты liveness и readiness проб <code>vm-route-forge</code> модуля <code>virtualization</code>
4106	TCP	Порт pprof компонента <code>vm-route-forge</code> модуля <code>virtualization</code> в debug-режиме
4107	TCP	Порты gRPC liveness и readiness проб <code>virtualization-dra</code> модуля <code>virtualization</code>

Внешний трафик к master-узлам

Порт	Протокол	Назначение
22	TCP	SSH для инициализации Deckhouse Kubernetes Platform
6443	TCP	Прямой доступ к API-серверу

Внешний трафик к фронтенд-узлам

Порт	Протокол	Назначение
80, 443	TCP	Прикладные порты для запросов к Ingress-контроллеру по протоколам HTTP и HTTPS. Обратите внимание, что эти порты настраиваются в ресурсе <code>IngressNginxController</code> и могут отличаться в разных инсталляциях
5416	UDP	OpenVPN
5416	TCP	OpenVPN
10256	TCP	Healthcheck-порт для внешних балансировщиков
30000-32767	TCP	Диапазон портов NodePort

Внешний трафик для всех узлов

Порт	Протокол	Назначение
53	UDP	DNS
53	TCP	DNS
123	UDP	NTP для синхронизации с внешними серверами точного времени
179	TCP	BGP-стык с роутером инфраструктуры для балансировки сетевого трафика (модуль <code>metallb</code>)

Порт	Протокол	Назначение
443	TCP	Container registry

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Параметры модуля `cni-cilium`
- Параметры модуля `cni-flannel`
- Параметры модуля `control-plane-manager`

Список алертов

На странице приведен список алертов мониторинга в Deckhouse Kubernetes Platform.



Список не содержит алертов подключаемых модулей.

Алерты сгруппированы по модулям. Справа от названия алерта указаны иконки минимальной редакции DKP в которой доступен алерт и уровня критичности алерта (severity).

Для каждого алерта приведено краткое описание (summary), раскрыв которое можно увидеть подробное описание алерта (description), при их наличии.

Критичность алерта

В описании алертов присутствует параметр Severity (S), означающий уровень критичности. Его значение варьируется от S1 до S9 и может расцениваться следующим образом:

- S1 — максимальный уровень, критический сбой/авария (требуется незамедлительные действия);
- S2 — высокий уровень, близкий к максимальному, возможная авария (необходимо быстрое реагирование);
- S3 — средний уровень, потенциально серьёзная проблема (необходима проверка);
- S4-S9 — низкий уровень. Есть проблема, но в целом работоспособность не нарушена.

Модуль admission-policy-engine

- D8AdmissionPolicyEngineNotBootstrapped CE S7
▼ **Admission-policy-engine module hasn't been bootstrapped for 10 minutes.**

The admission-policy-engine module couldn't bootstrap.

Steps to troubleshoot:

1. Verify that the module's components are up and running:

```
d8 k get pods -n d8-admission-policy-engine
```

2. Check logs for issues, such as missing constraint templates or incomplete CRD creation:

```
d8 k logs -n d8-system -lapp=deckhouse --tail=1000 | grep admission-policy-engine
```

- OperationPolicyViolation CE S3
▼ **At least one object violates the configured cluster operation policies.**

You have configured operation policies for the cluster, and one or more existing objects are violating these policies.

To identify violating objects:

- Run the following Prometheus query:

```
count by (violating_namespace, violating_kind, violating_name,
violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    violation_enforcement="deny",
    source_type="OperationPolicy"
  }
)
```

- Alternatively, check the admission-policy-engine Grafana dashboard.

- PodSecurityStandardsViolation CE S3
▼ **At least one pod violates the configured cluster pod security standards.**

You have configured Pod Security Standards, and one or more running pods are violating these standards.

To identify violating pods:

- Run the following Prometheus query:

```
count by (violating_namespace, violating_name, violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    violation_enforcement="deny",
    violating_namespace=~".*",
    violating_kind="Pod",
    source_type="PSS"
  }
)
```

- Alternatively, check the admission-policy-engine Grafana dashboard.

- SecurityPolicyViolation CE S3

- ▼ **At least one object violates the configured cluster security policies.**

You have configured security policies for the cluster, and one or more existing objects are violating these policies.

To identify violating objects:

- Run the following Prometheus query:

```
count by (violating_namespace, violating_kind, violating_name,
violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    violation_enforcement="deny",
    source_type="SecurityPolicy"
  }
)
```

- Alternatively, check the admission-policy-engine Grafana dashboard.

Модуль cert-manager

- CertmanagerCertificateChallengePending CE S4

- ▼ **ACME Challenge for domain "XXX" is pending for more than 1 hour.**

The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than 1 hour and requires investigation.

Common causes:

- DNS misconfiguration (for DNS-01 challenges).
- Ingress or firewall issues blocking HTTP-01 validation.
- ACME provider rate limiting.
- Network connectivity issues.

Resolution steps:

1. Check the Challenge status and reason:

```
d8 k -n NAMESPACE describe challenge NAME
```

2. Check `cert-manager` logs for errors:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -C cert-manager --tail=100
```

3. If the challenge is stuck, delete the Order to trigger a retry:

```
d8 k -n NAMESPACE get orders
d8 k -n NAMESPACE delete order ORDER_NAME
```

- CertmanagerCertificateChallengePending CE S6

- ▼ **ACME Challenge for domain "XXX" is pending for more than 10 minutes.**

The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than 10 minutes.

This may indicate issues with the ACME challenge validation process.

Common causes:

- DNS propagation delays (for DNS-01 challenges).
- Ingress or firewall issues blocking HTTP-01 validation.
- ACME provider rate limiting or temporary unavailability.

Diagnostic steps:

1. Check the Challenge status:

```
d8 k -n NAMESPACE describe challenge NAME
```

2. Check `cert-manager` logs for errors:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -C cert-manager --tail=100
```

- CertmanagerCertificateChallengeStuck CE S4

▼ ACME Challenge for domain "XXX" is stuck in "STATUS_REFERENCE" state.

The ACME Challenge `NAMESPACE/NAME` for domain `XXX` is in `STATUS_REFERENCE` state and requires manual intervention.

Reason: XXX

The certificate will not be issued or renewed until the issue is resolved.

Common causes:

- DNS misconfiguration (for DNS-01 challenges).
- Ingress or firewall issues (for HTTP-01 challenges).
- Rate limiting by the ACME provider.
- Authorization expired before challenge completion.

Resolution steps:

1. Check the Challenge status and reason:

```
d8 k -n NAMESPACE describe challenge NAME
```

2. Delete the failed Order to trigger a new certificate request:

```
d8 k -n NAMESPACE get orders
d8 k -n NAMESPACE delete order ORDER_NAME
```

- CertmanagerCertificateExpired CE S4

▼ Certificate `NAMESPACE/NAME` is not provided.

Certificate is not provided.

To check the certificate details, run the following command:

```
d8 k -n NAMESPACE describe certificate NAME
```

- CertmanagerCertificateExpiredSoon CE S4

▼ Certificate will expire soon.

The certificate `NAMESPACE/NAME` will expire in less than two weeks.

To check the certificate details, run the following command:

```
d8 k -n NAMESPACE describe certificate CERTIFICATE-NAME
```

- CertmanagerCertificateOrderErrors CE S5

▼ Cert-manager couldn't order a certificate.

Cert-manager received responses with the status code `STATUS_REFERENCE` when requesting `SCHEME://HOST_NAMEPATH`.

This can affect certificate ordering and prolongation in the future. For details, check the cert-manager logs using the following command:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -C cert-manager
```

Модуль chrony

- NodeTimeOutOfSync CE S5
▼ **Clock on the node `NODE_NAME` is drifting.**

Time on the node `NODE_NAME` is out of sync and drifts apart from the NTP server clock by `VALUE` seconds.

To resolve the time synchronization issues:

- Fix network errors:
 - Ensure the upstream time synchronization servers defined in the `chrony` configuration are available.
 - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
- Modify the NTP servers list defined in the `chrony` configuration.
- NTPDaemonOnNodeDoesNotSynchronizeTime CE S5
▼ **NTP daemon on the node `NODE_NAME` haven't synchronized time for too long.**

Steps to troubleshoot:

1. Check if the `chrony` pod is running on the node by executing the following command:

```
d8 k -n d8-chrony --field-selector spec.nodeName="NODE_NAME" get pods
```

2. Verify the `chrony` daemon's status by executing the following command:

```
d8 k -n d8-chrony exec POD_NAME -- /opt/chrony-static/bin/chronyc sources
```

3. Resolve the time synchronization issues:

- Fix network errors:
 - Ensure the upstream time synchronization servers defined in the `chrony` configuration are available.
 - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
- Modify the NTP servers list defined in the `chrony` configuration.

Модуль cloud-provider-huaweicloud

- D8HuaweiCloudMachineStuckInDeleting EE S6
▼ **HuaweiCloudMachine `NAME` is stuck in Deleting for more than 24h.**

The `HuaweiCloudMachine` `NAME` in namespace `NAMESPACE` has been stuck in Deleting state for more than 24 hours.

This usually means the deletion job in HuaweiCloud is failing or the API is unavailable.

To inspect the machine status, run:

```
d8 k get huaweicloudmachine -n NAMESPACE NAME -O yaml
```

Check the controller logs for errors:

```
d8 k logs -n d8-cloud-provider-huaweicloud -l app=caphc-controller-manager
```

Модуль cloud-provider-yandex

- D8YandexNatInstanceConnectionsQuotaUtilization CE S4
▼ **Connection quota utilization of the Yandex NAT instance exceeds 85% over the last 5 minutes.**

The connection quota for the Yandex NAT instance has exceeded 85% utilization over the past 5 minutes.

To prevent potential issues, contact Yandex technical support and request an increase in the

connection quota.

- NATInstanceWithDeprecatedAvailabilityZone CE S9
 - ▼ **NAT instance NAME is in a deprecated availability zone.**

The NAT instance `NAME` is located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud. To resolve this issue, migrate the NAT instance to either `ru-central1-a` or `ru-central1-b` by following the instructions below.

i The migration process involves irreversible changes and may result in a significant downtime. The duration typically depends on Yandex Cloud's response time and can last several tens of minutes.

1. Migrate the NAT instance. To get `providerClusterConfiguration.withNATInstance`, run the following command:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller module
values -g cloud-provider-yandex -O json | jq -C | jq
'.cloudProviderYandex.internal.providerClusterConfiguration.withNATInstance'
```

- If you specified `withNATInstance.natInstanceInternalAddress` and/or `withNATInstance.internalSubnetID` in `providerClusterConfiguration`, remove them using the following command:

```
d8 system edit provider-cluster-configuration
```

- If you specified `withNATInstance.externalSubnetID` and/or `withNATInstance.natInstanceExternalAddress` in `providerClusterConfiguration`, change them to the appropriate values.

To get the address and subnet ID, use the Yandex Cloud Console or CLI.

To change `withNATInstance.externalSubnetID` and `withNATInstance.natInstanceExternalAddress`, run the following command:

```
d8 system edit provider-cluster-configuration
```

2. Run the appropriate edition and version of the Deckhouse installer container on the **local** machine. You may have to change the container registry address to do that. After that, perform the converge.

- Get the appropriate edition and version of Deckhouse:

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -O
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -O
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr
'[:upper:]' '[:lower:]')
echo "DH_VERSION=$DH_VERSION DH_EDITION=$DH_EDITION"
```

- Run the installer:

```
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/"
registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

- Perform the converge:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/SSH_KEY_FILENAME --ssh-
user=USERNAME --ssh-host MASTER-NODE-0-HOST
```

3. Update the route table.

- Get the route table name:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller
module values -g cloud-provider-yandex -O json | jq -C | jq
'.global.clusterConfiguration.cloud.prefix'
```

- Get the NAT instance name:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- deckhouse-controller
module values -g cloud-provider-yandex -O json | jq -C | jq
'.cloudProviderYandex.internal.providerDiscoveryData.natInstanceName'
```

- Get the NAT instance internal IP address:

```
yc compute instance list | grep -e "INTERNAL IP" -e
NAT_INSTANCE_NAME_FROM_PREVIOUS_STEP
```

- Update the route:

```
yc vpc route-table update --name ROUTE_TABLE_NAME_FROM_PREVIOUS_STEP --route
"destination=0.0.0.0/0,next-
hop=NAT_INSTANCE_INTERNAL_IP_FROM_PREVIOUS_STEP"
```

- NodeGroupNodeWithDeprecatedAvailabilityZone CE S9
▼ **NodeGroup `NODE_GROUP_NAME` contains nodes in a deprecated availability zone.**

Certain nodes in the node group `NODE_GROUP_NAME` are located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud.

Steps to troubleshoot:

1. Identify the nodes that need to be migrated by running the following command:

```
d8 k get node -l "topology.kubernetes.io/zone=ru-central1-c"
```

2. Migrate your nodes, disks, and load balancers to one of the supported zones: `ru-central1-a`, `ru-central1-b`, or `ru-central1-d`. Refer to the Yandex migration guide for detailed instructions.

i You can't migrate public IP addresses between zones. For details, refer to the migration guide.

Модуль `cni-cilium`

- CiliumAgentEndpointsNotReady CE S4
▼ **Over 50% of all known endpoints aren't ready in agent `NAMESPACE/POD_NAME`.**

For details, refer to the logs of the agent:

```
d8 k -n NAMESPACE logs POD_NAME
```

- CiliumAgentMapPressureCritical CE S4
▼ **eBPF map `MAP_NAME` exceeds 90% utilization in agent `NAMESPACE/POD_NAME`.**

The eBPF map resource utilization limit has almost been reached.

Check with the vendor for potential remediation steps.

- CiliumAgentMetricNotFound CE S4
▼ **Agent `NAMESPACE/POD_NAME` isn't sending some metrics.**

Steps to troubleshoot:

1. Check the logs of the agent:

```
d8 k -n NAMESPACE logs POD_NAME
```

2. Verify the agent's health status:

```
d8 k -n NAMESPACE exec -ti POD_NAME cilium-health status
```

3. Compare the metrics with those of a neighboring agent.



Note that the absence of metrics can indirectly indicate that new pods can't be created on the node due to connectivity issues with the agent.

- CiliumAgentPolicyImportErrors CE S4
▼ **Agent NAMESPACE/POD_NAME fails to import policies.**

For details, refer to the logs of the agent:

```
d8 k -n NAMESPACE logs POD_NAME
```

- CiliumAgentUnreachableHealthEndpoints CE S4
▼ **Agent NAMESPACE/POD_NAME can't reach some of the node health endpoints.**

For details, query Cilium health status API:

```
d8 k -n NAMESPACE exec POD_NAME -- cilium-health status
```

You can also refer to the agent logs:

```
d8 k -n NAMESPACE logs POD_NAME
```

- CniCiliumNonStandardVXLANPortFound CE S4
▼ **Cilium configuration uses a non-standard VXLAN port.**

The Cilium configuration specifies a non-standard VXLAN port `XXX`. The recommended port number is `XXX`.

To resolve this issue, update the `tunnel-port` parameter in the `cilium-configmap` ConfigMap located in the `d8-cni-cilium` namespace to match the recommended port.

If you configured the non-standard port on purpose, ignore this alert.

- CniCiliumOrphanEgressGatewayPolicyFound SE-PLUS S4
▼ **Orphaned EgressGatewayPolicy with an irrelevant EgressGateway name has been found.**

The cluster contains an orphaned EgressGatewayPolicy named `NAME` with an irrelevant EgressGateway name.

To resolve this issue, verify the EgressGateway name specified in the EgressGatewayPolicy resource `EGRESSGATEWAY` and update it as needed.

- D8NodeHasUnmetKernelRequirements CE S4
▼ **Nodes have unmet kernel requirements.**

Deckhouse has detected that some nodes don't meet the required kernel constraints. As a result, certain modules can't run on these nodes.

Kernel requirements for each module:

- **Cilium module:** Kernel version must be `>= 5.8`.

To list all affected nodes, use the `d8_node_kernel_does_not_satisfy_requirements == 1` expression in Prometheus.

Модуль control-plane-manager

- D8ControlPlaneManagerPodNotRunning CE S6
▼ **Controller pod isn't running on node NODE_NAME.**

The `d8-control-plane-manager` pod is either failing or hasn't been scheduled on node `NODE_NAME`.

To resolve this issue, check the status of the `kube-system/d8-control-plane-manager`

DaemonSet and its pods by running the following command:

```
d8 k -n kube-system get daemonset,pod --selector=app=d8-control-plane-manager
```

- D8EtcdDatabaseHighFragmentationRatio CE S7
▼ **etcd database size in use is less than 50% of the allocated storage.**

The etcd database size in use on instance `INSTANCE_NAME` is less than 50% of the allocated disk space, indicating potential fragmentation. Additionally, the total storage size exceeds 75% of the configured quota.

To resolve this issue, defragment the etcd database by running the following command:

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

- D8EtcdExcessiveDatabaseGrowth CE S4
▼ **etcd cluster database is growing rapidly.**

Based on the growth rate observed over the last six hours, Deckhouse predicts that the etcd database will run out of disk space within one day on instance `INSTANCE_NAME`.

To prevent disruptions, investigate the cause and take necessary action.

- D8EtcdHighRequestsLatency CE S5
▼ **Etcd request latency (p99) has exceeded 800 ms multiple times in the past hour.**

The 99th percentile of etcd request duration on `INSTANCE_NAME` has exceeded 800 ms more than twice in the last hour.

High etcd latency can cause API server timeouts, slow admission webhooks, and general cluster instability.

Investigate the following:

- **Node disk I/O:** Etcd is extremely sensitive to disk performance. Check disk utilization and I/O wait on the control plane node.
- **Network-attached storage:** If the etcd data directory resides on NFS, Ceph, or another network storage, check connectivity and latency to the storage backend. Etcd requires a consistently low-latency local disk (SSD recommended).
- **Control plane summary:** Review the **Kubernetes Cluster → Control Plane Status** Grafana dashboard for an overall picture of API server and etcd health, request rates, and error rates.
- **Node base performance:** Check node CPU and RAM utilization and load average. If the node is CPU-bound (load average significantly exceeds the number of CPU cores), consider scaling up the control plane node.
- D8KubeEtcdDatabaseSizeCloseToTheLimit CE S3
▼ **etcd database size is approaching the limit.**

The etcd database size on `NODE_NAME` is nearing its size limit. This may be caused by a high number of events, such as pod evictions or the recent creation of numerous resources in the cluster.

Possible solutions:

- Defragment the etcd database by running the following command:

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

- Increase node memory. Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
...	...

- D8KubernetesStaleTokensDetected CE S8

- ▼ **Stale service account tokens detected for XXX/XXX.**

Service account `XXX/XXX` is using a stale token. This issue may occur if an application reads the token only at startup and does not reload it periodically. As a result, an outdated token might be used, leading to security breach and authentication failures.

Recommended actions:

- Ensure your application is configured to periodically reload the token from the file system.
- Verify that you are using an up-to-date client library that supports automatic token rotation.

Note that currently these tokens are not blocked because the `--service-account-extend-token-expiration` flag is enabled by default (Default: `true`). With this flag enabled, admission-injected tokens are extended up to 1 year during token generation to facilitate a safe transition from legacy tokens to the bound service account token feature, ignoring the value of `service-account-max-token-expiration`.

For further investigation: You can find the problematic pods using the following command:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.serviceAccountName=="XXX" and .metadata.namespace=="XXX") | .metadata.name'
```

- D8KubernetesVersionIsDeprecated CE S7

- ▼ **Kubernetes version VERSION_NUMBER is deprecated.**

The current Kubernetes version `VERSION_NUMBER` has been deprecated, and support for it will be removed in upcoming releases.

Please migrate to the next kubernetes version (at least 1.32)

Read documentation about how to update the Kubernetes version in the cluster.

- D8NeedDecreaseEtcdQuotaBackendBytes CE S6

- ▼ **Deckhouse suggests reducing quota-backend-bytes.**

When the control plane node memory is reduced, Deckhouse may suggest reducing `quota-backend-bytes`. While Deckhouse is capable of automatically increasing this value, reducing it must be done manually.

To modify `quota-backend-bytes`, set the `controlPlaneManager.etcd.maxDbSize` parameter. Before setting a new value, check the current database usage on every control plane node by running:

```
for pod in $(d8 k get pod -n kube-system -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec -ti "$pod" -- /usr/bin/etcdctl \
    --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt \
    --key /etc/kubernetes/pki/etcd/ca.key \
    endpoint status -w json | jq --arg a "$pod" -r \
    '[0].Status.dbSize / 1024 / 1024 | tostring | $a + ": " + . + " MB"';
done
```

Things to note:

- The maximum value for `controlPlaneManager.etcd.maxDbSize` is 8 GB.
- If control plane nodes have less than 24 GB, set `controlPlaneManager.etcd.maxDbSize` to

2 GB.

- Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
...	...

- D8ProblematicFeatureGateInUse CE S9

▼ **Feature gate XXX is STATUS_REFERENCE in current Kubernetes version XXX and should be removed from the configuration. Is not using in the cluster control-plane.**

The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it is STATUS_REFERENCE in the current Kubernetes version `XXX`.

XXX This feature gate does not exist in any component (apiserver, kube-controller-manager, kube-scheduler, kubelet) and is not being applied to the cluster. XXX

Recommended actions:

- Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

```
d8 k edit mc control-plane-manager
```

For more information about Kubernetes feature gates, refer to the [official documentation](#).

- D8UsingFeatureGateWillBeDeprecated CE S9

▼ **Feature gate XXX will be deprecated in Kubernetes version XXX.**

The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it will be deprecated in Kubernetes version `XXX`.

Current Kubernetes version: `XXX`

Recommended actions:

- Review the usage of this feature gate in your cluster.
- Plan migration away from this feature gate before upgrading to version `XXX`.
- Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

```
d8 k edit mc control-plane-manager
```

For more information about Kubernetes feature gates, refer to the [official documentation](#).

Модуль documentation

- ModuleConfigDeprecated CE S9

▼ **Deprecated deckhouse-web ModuleConfig detected.**

The `deckhouse-web` module has been renamed to `documentation`, and a new `documentation` ModuleConfig is generated automatically.

Steps to troubleshoot:

- Remove the deprecated `deckhouse-web` ModuleConfig from the CI deployment process.

2. Delete it using the following command:

```
d8 k delete mc deckhouse-web
```

- ModuleDocumentationRenderError CE S6

- ▼ **Documentation rendering failed for module MODULE_NAME.**

An error occurred while rendering documentation for the `MODULE_NAME` module. The broken module was removed from the documentation site during Hugo build.

Check the `ModuleDocumentation` resource status for details:

```
d8 k get moduledocumentations MODULE_NAME -O yaml
```

Check docs-builder logs for more information:

```
d8 k -n d8-system logs -l app=docs-builder | grep -i "removed broken module"
```

Модуль extended-monitoring

- CertificateSecretExpired CE S8

- ▼ **Certificate has expired.**

A certificate in Secret `SECRET_NAMESPACE/SECRET_NAME` has expired.

Ways to resolve:

- If the certificate is managed manually, upload a new certificate.
- If the certificate is managed by the `cert-manager` module, inspect the certificate resource:

1. Retrieve the certificate name from the Secret:

```
cert=$(d8 k get secret -n SECRET_NAMESPACE SECRET_NAME -o 'jsonpath={.metadata.annotations.cert-manager\.io/certificate-name}')
```

2. Check the certificate status and investigate why it hasn't been updated:

```
d8 k describe cert -n SECRET_NAMESPACE "$cert"
```

- CertificateSecretExpiredSoon CE S8

- ▼ **Certificate is expiring soon.**

A certificate in Secret `SECRET_NAMESPACE/SECRET_NAME` will expire in less than two weeks.

Ways to resolve:

- If the certificate is managed manually, upload a new certificate.
- If the certificate is managed by the `cert-manager` module, inspect the certificate resource:

1. Retrieve the certificate name from the Secret:

```
cert=$(d8 k get secret -n SECRET_NAMESPACE SECRET_NAME -o 'jsonpath={.metadata.annotations.cert-manager\.io/certificate-name}')
```

2. Check the certificate status and investigate why it hasn't been updated:

```
d8 k describe cert -n SECRET_NAMESPACE "$cert"
```

- CronJobAuthenticationFailure CE S7

- ▼ **Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.**

Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.

- The `CONTAINER_NAME` container in the registry.
- CronJobAuthorizationFailure CE S7
▼ **Insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.**

Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.
- The `CONTAINER_NAME` container in the registry.
- CronJobBadImageFormat CE S7
▼ **The `IMAGE_NAME` image name is incorrect.**

Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.
- The `CONTAINER_NAME` container in the registry.
- CronJobFailed CE S5
▼ **Job `NAMESPACE/JOB_NAME` failed in CronJob `NAMESPACE/CRONJOB`.**

Deckhouse has detected that Job `NAMESPACE/JOB_NAME` failed in CronJob `NAMESPACE/CRONJOB`.

Steps to resolve:

1. Print the job details:

```
d8 k -n NAMESPACE describe job JOB_NAME
```

2. Check the job status:

```
d8 k -n NAMESPACE get job JOB_NAME
```

3. Check the status of pods created by the job:

```
d8 k -n NAMESPACE get pods -l job-name=JOB_NAME
```

- CronJobImageAbsent CE S7
▼ **The `IMAGE_NAME` image is missing from the registry.**

Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.
- The `CONTAINER_NAME` container in the registry.
- CronJobImageAvailabilityUnknownError CE S7
▼ **An unknown error occurred with the `IMAGE_NAME` image availability check.**

Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.
- The `CONTAINER_NAME` container in the registry.

Review the image-availability-exporter logs to see the detailed error information:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- CronJobPodsNotCreated CE S5
▼ **Pods set in CronJob `NAMESPACE/JOB_NAME` haven't been created.**

Deckhouse has detected that the pods set in CronJob `NAMESPACE/CRONJOB` still haven't been created.

Steps to resolve:

1. Print the job details:

```
d8 k -n NAMESPACE describe job JOB_NAME
```

2. Check the job status:

```
d8 k -n NAMESPACE get job JOB_NAME
```

3. Check the status of pods created by the job:

```
d8 k -n NAMESPACE get pods -l job-name=JOB_NAME
```

- CronJobRegistryUnavailable CE S7

▼ **The container registry is not available for the IMAGE_NAME image.**

Deckhouse has detected that the container registry is not available for the IMAGE_NAME image.

To resolve this issue, investigate the possible causes in the following sources:

- The NAMESPACE namespace.
- The CronJob NAME.
- The CONTAINER_NAME container in the registry.

- CronJobSchedulingError CE S6

▼ **CronJob NAMESPACE/CRONJOB failed to schedule on time.**

Deckhouse has detected that CronJob NAMESPACE/CRONJOB failed to schedule on time.

- Current schedule: XXX
- Last scheduled time: XXX%
- Next projected schedule time: XXX%

- D8CertExporterPodIsNotReady CE S8

▼ **The x509-certificate-exporter pod isn't ready.**

Steps to resolve:

1. Retrieve the deployment details:

```
d8 k -n d8-monitoring describe deploy x509-certificate-exporter
```

2. Check the pod status and investigate why it's not ready:

```
d8 k -n d8-monitoring describe pod -l app=x509-certificate-exporter
```

- D8CertExporterPodIsNotRunning CE S8

▼ **The x509-certificate-exporter pod isn't running.**

Steps to resolve:

1. Retrieve the deployment details:

```
d8 k -n d8-monitoring describe deploy x509-certificate-exporter
```

2. Check the pod status and investigate why it's not running:

```
d8 k -n d8-monitoring describe pod -l app=x509-certificate-exporter
```

- D8CertExporterTargetAbsent CE S8

▼ **There is no x509-certificate-exporter target in Prometheus.**

Ways to resolve:

- Check the pod status:

```
d8 k -n d8-monitoring get pod -l app=x509-certificate-exporter
```

- Check the pod logs:

```
d8 k -n d8-monitoring logs -l app=x509-certificate-exporter -C x509-certificate-exporter
```

- D8CertExporterTargetDown CE S8
▼ **Prometheus can't scrape x509-certificate-exporter metrics.**

Ways to resolve:

- Check the pod status:

```
d8 k -n d8-monitoring get pod -l app=x509-certificate-exporter
```

- Check the pod logs:

```
d8 k -n d8-monitoring logs -l app=x509-certificate-exporter -C x509-certificate-exporter
```

- D8ImageAvailabilityExporterMalfunctioning CE S8
▼ **The image-availability-exporter has crashed.**

The `image-availability-exporter` has failed to perform any image availability checks in the container registry for over 20 minutes.

To investigate the issue, review the exporter's logs:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- D8ImageAvailabilityExporterPodIsNotReady CE S8
▼ **The image-availability-exporter pod is not ready.**

Deckhouse has detected that the `image-availability-exporter` pod is not ready. As a result, the images listed in the `image` field aren't checked for availability in the container registry.

Steps to resolve:

1. Retrieve the deployment details:

```
d8 k -n d8-monitoring describe deploy image-availability-exporter
```

2. Check the pod status and investigate why it isn't `Ready`:

```
d8 k -n d8-monitoring describe pod -l app=image-availability-exporter
```

- D8ImageAvailabilityExporterPodIsNotRunning CE S8
▼ **The image-availability-exporter pod is not running.**

Deckhouse has detected that the `image-availability-exporter` pod is not running. As a result, the images listed in the `image` field aren't checked for availability in the container registry.

Steps to resolve:

1. Retrieve the deployment details:

```
d8 k -n d8-monitoring describe deploy image-availability-exporter
```

2. Check the pod status and investigate why it isn't running:

```
d8 k -n d8-monitoring describe pod -l app=image-availability-exporter
```

- D8ImageAvailabilityExporterTargetAbsent CE S8
▼ **The image-availability-exporter target is missing from Prometheus.**

Deckhouse has detected that the `image-availability-exporter` target is missing from Prometheus.

Steps to resolve:

1. Check the pod status:

```
d8 k -n d8-monitoring get pod -l app=image-availability-exporter
```

2. Check the pod logs:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- D8ImageAvailabilityExporterTargetDown CE S8
▼ **Prometheus can't scrape metrics of image-availability-exporter.**

Deckhouse has detected that Prometheus is unable to scrape metrics of `image-availability-exporter`.

Steps to resolve:

1. Check the pod status:

```
d8 k -n d8-monitoring get pod -l app=image-availability-exporter
```

2. Check the pod logs:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- DaemonSetAuthenticationFailure CE S7
▼ **Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.**

Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The DaemonSet `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DaemonSetAuthorizationFailure CE S7
▼ **Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.**

Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
 - The DaemonSet `NAME`.
 - The `CONTAINER_NAME` container in the registry.
 - DaemonSetBadImageFormat CE S7
▼ **The IMAGE_NAME image name is incorrect.**
- Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.
- To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

- The `NAMESPACE` namespace.
- The DaemonSet `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DaemonSetImageAbsent CE S7
▼ **The IMAGE_NAME image is missing from the registry.**

Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

- The `NAMESPACE` namespace.
- The DaemonSet `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DaemonSetImageAvailabilityUnknownError CE S7
▼ **An unknown error occurred with the IMAGE_NAME image availability check.**

Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

- The `NAMESPACE` namespace.

- The DaemonSet `NAME`.
- The `CONTAINER_NAME` container in the registry.

Review the image-availability-exporter logs to see the detailed error information:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- DaemonSetRegistryUnavailable CE S7

▼ **The container registry is not available for the `IMAGE_NAME` image.**

Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The DaemonSet `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DeploymentAuthenticationFailure CE S7
- ▼ **Unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.**

Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The Deployment `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DeploymentAuthorizationFailure CE S7
- ▼ **Insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.**

Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The Deployment `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DeploymentBadImageFormat CE S7
- ▼ **The `IMAGE_NAME` image name is incorrect.**

Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

- The `NAMESPACE` namespace.
- The Deployment `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DeploymentImageAbsent CE S7
- ▼ **The `IMAGE_NAME` image is missing from the registry.**

Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

- The `NAMESPACE` namespace.
- The Deployment `NAME`.
- The `CONTAINER_NAME` container in the registry.
- DeploymentImageAvailabilityUnknownError CE S7
- ▼ **An unknown error occurred with the `IMAGE_NAME` image availability check.**

Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

- The `NAMESPACE` namespace.
- The Deployment `NAME`.
- The `CONTAINER_NAME` container in the registry.

Review the image-availability-exporter logs to see the detailed error information:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- DeploymentRegistryUnavailable CE S7
▼ **The container registry is not available for the IMAGE_NAME image.**

Deckhouse has detected that the container registry is not available for the IMAGE_NAME image.

To resolve this issue, investigate the possible causes in the following sources:

- The NAMESPACE namespace.
- The Deployment NAME.
- The CONTAINER_NAME container in the registry.
- ExtendedMonitoringTargetDown CE S5
▼ **Extended monitoring is unavailable.**

The pod running extended-monitoring-exporter is currently unavailable.

As a result, the following alerts will not be triggered:

- Low disk space and inode usage on volumes.
- CPU overloads and container throttling.
- 500 errors on Ingress.
- Insufficient replicas of Deployments, StatefulSets, and DaemonSets.
- Other alerts associated with this exporter.

To resolve this issue, investigate its possible causes:

1. Print detailed information about the extended-monitoring-exporter deployment:

```
d8 k -n d8-monitoring describe deploy extended-monitoring-exporter
```

2. Print detailed information about the pods associated with the extended-monitoring-exporter:

```
d8 k -n d8-monitoring describe pod -l app=extended-monitoring-exporter
```

- IngressResponses5xx CE S4
▼ **URL VHOST/LOCATION on Ingress INGRESS has more than XXX% of 5xx responses from the backend.**

Deckhouse has detected that URL VHOST/LOCATION on Ingress INGRESS, using service SERVICE_NAME on port SERVICE_PORT has more than XXX% of 5xx responses from the backend.

Current rate of 5xx responses: VALUE%

- IngressResponses5xx CE S5
▼ **URL VHOST/LOCATION on Ingress INGRESS has more than XXX% of 5xx responses from the backend.**

Deckhouse has detected that URL VHOST/LOCATION on Ingress INGRESS, using service SERVICE_NAME on port SERVICE_PORT, has more than XXX% of 5xx responses from the backend.

Current rate of 5xx responses: VALUE%

- KubernetesDaemonSetNotUpToDate CE S9
▼ **There were VALUE outdated pods in DaemonSet NAMESPACE/DAEMONSET_NAME over the last 15 minutes.**

Deckhouse has detected VALUE outdated pods in DaemonSet NAMESPACE/DAEMONSET_NAME over the last 15 minutes.

Steps to resolve:

1. Check the DaemonSet's status:

```
d8 k -n NAMESPACE get ds DAEMONSET_NAME
```

2. Analyze the DaemonSet's description:

```
d8 k -n NAMESPACE describe ds DAEMONSET_NAME
```

3. If the parameter Number of Nodes Scheduled with Up-to-date Pods does not match Current Number of Nodes Scheduled, check the DaemonSet's updateStrategy:

```
d8 k -n NAMESPACE get ds DAEMONSET_NAME -o json | jq '.spec.updateStrategy'
```

If `updateStrategy` is set to `OnDelete`, the DaemonSet is updated only when pods are deleted.

- KubernetesDaemonSetReplicasUnavailable CE S5
▼ **No available replicas remaining in DaemonSet NAMESPACE/DAEMONSET_NAME.**

Deckhouse has detected that there are no available replicas remaining in DaemonSet `NAMESPACE/DAEMONSET_NAME`.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

```
d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name == "DAEMONSET_NAME")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
"Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
SelectorTerms[].matchFields[].values[]'
```

- KubernetesDaemonSetReplicasUnavailable CE S6
▼ **The number of unavailable replicas in DaemonSet NAMESPACE/DAEMONSET_NAME exceeds the threshold.**

Deckhouse has detected that the number of unavailable replicas in DaemonSet `NAMESPACE/DAEMONSET_NAME` exceeds the threshold.

- Current number: `VALUE` unavailable replica(s).
- Threshold number: `XXX` unavailable replica(s).

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

```
d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name == "DAEMONSET_NAME")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
"Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
SelectorTerms[].matchFields[].values[]'
```

- KubernetesDeploymentReplicasUnavailable CE S5
▼ **No available replicas remaining in deployment NAMESPACE/DEPLOYMENT_NAME.**

Deckhouse has detected that there are no available replicas remaining in deployment `NAMESPACE/DEPLOYMENT_NAME`.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- KubernetesDeploymentReplicasUnavailable CE S6
▼ **The number of unavailable replicas in deployment NAMESPACE/DEPLOYMENT_NAME exceeds spec.strategy.rollingupdate.maxunavailable.**

Deckhouse has detected that the number of unavailable replicas in deployment `NAMESPACE/DEPLOYMENT NAME` exceeds the value set in `spec.strategy.rollingupdate.maxunavailable`.

- Current number: `VALUE` unavailable replica(s).
- Threshold number: `XXX` unavailable replica(s).

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- KubernetesStatefulSetReplicasUnavailable CE S5
▼ **No ready replicas remaining in StatefulSet NAMESPACE/STATEFULSET.**

Deckhouse has detected that there are no ready replicas remaining in StatefulSet NAMESPACE/STATEFULSET.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- KubernetesStatefulSetReplicasUnavailable CE S6
▼ **The number of unavailable replicas in StatefulSet NAMESPACE/STATEFULSET exceeds the threshold.**

Deckhouse has detected that the number of unavailable replicas in StatefulSet NAMESPACE/STATEFULSET exceeds the threshold.

- Current number: VALUE unavailable replica(s).
- Threshold number: XXX unavailable replica(s).

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- LoadAverageHigh CE S4
▼ **Average load on node NODE_NAME is too high.**

Over the last 5 minutes, the average load on node NODE_NAME has been higher than XXX per core.

There are more processes in the queue than the CPU can handle.

Possible causes:

- A process has created too many threads or child processes.
- The CPU is overloaded.

- LoadAverageHigh CE S5
▼ **Average load on node NODE_NAME is too high.**

Over the last 30 minutes, the average load on node NODE_NAME has been higher than or equal to XXX per core.

There are more processes in the queue than the CPU can handle.

Possible causes:

- A process has created too many threads or child processes.
- The CPU is overloaded.

- NodeDiskBytesUsage CE S5
▼ **Node disk NODE_DISK_NAME on mount point MOUNTPPOINT is using more than XXX% of its storage capacity.**

Deckhouse has detected that node disk NODE_DISK_NAME on mount point MOUNTPPOINT is using more than XXX% of its storage capacity.

Current storage usage: VALUE%

Steps to resolve:

1. Retrieve disk usage information on the node:

```
ncdu -X MOUNTPPOINT
```

2. If the output shows high disk usage in the /var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/ directory, identify the pods with the highest usage:

```
crictl stats -O json | jq '.stats[] | select((.writableLayer.usedBytes.value | tonumber) > 1073741824) | { meta: .attributes.labels, diskUsage: ((.writableLayer.usedBytes.value | tonumber) / 1073741824 * 100 | round / 100 | tostring + " GiB") }'
```

- NodeDiskBytesUsage CE S6
▼ **Node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.**

Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.

Current storage usage: VALUE%

Steps to resolve:

1. Retrieve disk usage information on the node:

```
ncdu -X MOUNTPPOINT
```

2. If the output shows high disk usage in the `/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/` directory, identify the pods with the highest usage:

```
crictl stats -O json | jq '.stats[] | select((.writableLayer.usedBytes.value | tonumber) > 1073741824) | { meta: .attributes.labels, diskUsage: ((.writableLayer.usedBytes.value | tonumber) / 1073741824 * 100 | round / 100 | tostring + " GiB") }'
```

- NodeDiskNodesUsage CE S5
▼ **Node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.**

Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.

Current storage usage: VALUE%

- NodeDiskNodesUsage CE S6
▼ **Node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.**

Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPPOINT` is using more than XXX% of its storage capacity.

Current storage usage: VALUE%

- PersistentVolumeClaimBytesUsage CE S4
▼ **PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.**

Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.

Current volume storage usage: VALUE%

PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- PersistentVolumeClaimBytesUsage CE S5
▼ **PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.**

Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.

Currently volume storage usage: VALUE%

PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

XXXXXX, XXXPOD_NAMEXXX

- PersistentVolumeClaimInodesUsed CE S4
▼ **PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.**

Deckhouse has detected that PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.

Current volume inode usage: VALUE%

PersistentVolumeClaim NAMESPACE/PVC_NAME is used by the following pods:

XXXXXX, XXXPOD_NAMEXXX

- PersistentVolumeClaimInodesUsed CE S5
▼ **PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.**

Deckhouse has detected that PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.

Current volume inode usage: VALUE%

PersistentVolumeClaim NAMESPACE/PVC_NAME is used by the following pods:

XXXXXX, XXXPOD_NAMEXXX

- StatefulSetAuthenticationFailure CE S7
▼ **Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.**

Deckhouse was unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.

To resolve this issue, investigate the possible causes in the following sources:

- The NAMESPACE namespace.
 - The StatefulSet NAME .
 - The CONTAINER_NAME container in the registry.
 - StatefulSetAuthorizationFailure CE S7
▼ **Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.**
- Deckhouse has insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.

To resolve this issue, investigate the possible causes in the following sources:

- The NAMESPACE namespace.
 - The StatefulSet NAME .
 - The CONTAINER_NAME container in the registry.
 - StatefulSetBadImageFormat CE S7
▼ **The IMAGE_NAME image name is incorrect.**
- Deckhouse has detected that the IMAGE_NAME image name is incorrect.
- To resolve this issue, check that the IMAGE_NAME image name is spelled correctly in the following sources:

- The NAMESPACE namespace.
- The StatefulSet NAME .
- The CONTAINER_NAME container in the registry.
- StatefulSetImageAbsent CE S7
▼ **The IMAGE_NAME image is missing from the registry.**

Deckhouse has detected that the IMAGE_NAME image is missing from the container registry.

To resolve this issue, check whether the IMAGE_NAME image is available in the following sources:

- The NAMESPACE namespace.
- The StatefulSet NAME .
- The CONTAINER_NAME container in the registry.
- StatefulSetImageAvailabilityUnknownError CE S7
▼ **An unknown error occurred with the IMAGE_NAME image availability check.**

Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

- The `NAMESPACE` namespace.
- The StatefulSet `NAME`.
- The `CONTAINER_NAME` container in the registry.

Review the image-availability-exporter logs to see the detailed error information:

```
d8 k -n d8-monitoring logs -l app=image-availability-exporter -C image-availability-exporter
```

- StatefulSetRegistryUnavailable CE S7
▼ **The container registry is not available for the `IMAGE_NAME` image.**

Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The StatefulSet `NAME`.
- The `CONTAINER_NAME` container in the registry.

Модуль ingress-nginx

- D8NginxIngressKruiseControllerPodsRestartingTooOften CE S8
▼ **Too many Kruise controller restarts detected.**

VALUE Kruise controller restarts detected in the last hour.

Excessive Kruise controller restarts indicate that something is wrong. Normally, it should be up and running all the time.

Steps to resolve:

1. Check events associated with `kruise-controller-manager` in the `d8-ingress-nginx` namespace. Look for issues related to node failures or memory shortages (OOM events):

```
d8 k -n d8-ingress-nginx get events | grep kruise-controller-manager
```

2. Analyze the controller's pod descriptions to identify restarted containers and possible causes. Pay attention to exit codes and other details:

```
d8 k -n d8-ingress-nginx describe pod -lapp=kruise,control-plane=controller-manager
```

3. In case the `kruise` container has restarted, get a list of relevant container logs to identify any meaningful errors:

```
d8 k -n d8-ingress-nginx logs -lapp=kruise,control-plane=controller-manager -C kruise
```

- GeolPDownloadErrorDetectedFromMaxMind CE S4
▼ **GeoIP DB download error in `JOB_NAME` (pod `POD_NAME`), edition `XXX`.**

The component `JOB_NAME` (pod `POD_NAME`) failed to download the GeoIP database from MaxMind for edition `XXX`. If this issue persists, investigate network connectivity or database availability.

- NginxIngressConfigTestFailed CE S4
▼ **Configuration test failed on Ingress `NGINX CONTROLLER_NAMESPACE/CONTROLLER_NAME`.**

The configuration test (`nginx -t`) for the `CONTROLLER_NAME` Ingress controller in the `CONTROLLER_NAMESPACE` namespace has failed.

Steps to resolve:

1. Check the controller logs:

```
d8 k -n CONTROLLER_NAMESPACE logs CONTROLLER_POD_NAME -C controller
```

2. Find the most recently created Ingress in the cluster:

```
d8 k get ingress --all-namespaces --sort-by="metadata.creationTimestamp"
```

3. Check for errors in the `configuration-snippet` or `server-snippet` annotations.

- NginxIngressDaemonSetNotUpToDate CE S9

▼ **There were VALUE outdated pods in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME over the last 20 minutes.**

Deckhouse has detected VALUE outdated pods in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME over the last 20 minutes.

Steps to resolve:

1. Check the DaemonSet's status:

```
d8 k -n NAMESPACE get ads DAEMONSET_NAME
```

2. Analyze the DaemonSet's description:

```
d8 k -n NAMESPACE describe ads DAEMONSET_NAME
```

3. If the parameter Number of Nodes Scheduled with Up-to-date Pods does not match Current Number of Nodes Scheduled, check the 'nodeSelector' and 'toleration' settings of the corresponding Ingress NGINX Controller and compare them to the 'labels' and 'taints' settings of the relevant nodes.

- NginxIngressDaemonSetReplicasUnavailable CE S4

▼ **No available replicas remaining in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME.**

Deckhouse has detected that there are no available replicas remaining in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

```
d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name == "DAEMONSET_NAME")) |
select(.status.phase != "Running" or ([.status.conditions[] | select(.type ==
"Ready" and .status == "False")] | length) == 1) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
SelectorTerms[].matchFields[].values[]'
```

- NginxIngressDaemonSetReplicasUnavailable CE S6

▼ **Some replicas of Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME are unavailable.**

Deckhouse has detected that some replicas of Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME are unavailable.

Current number: VALUE unavailable replica(s).

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

```
d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name == "DAEMONSET_NAME")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
"Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
SelectorTerms[].matchFields[].values[]'
```

- NginxIngressMaxMindAccountIDNotSet CE S6
▼ **MaxMind Account ID is missing for the Ingress NGINX Controller "XXX".**

The controller “XXX” downloads the GeoIP database using the legacy flow because `spec.geoIP2.maxmindAccountID` is not set. This fallback is less reliable and may be throttled by MaxMind. Specify both `maxmindAccountID` and `maxmindLicenseKey` in the corresponding `ingressnginxcontroller` resource to use the official GeoIP update client.

- NginxIngressPodIsRestartingTooOften CE S4
▼ **Too many Ingress NGINX restarts detected.**

VALUE Ingress NGINX Controller restarts detected in the last hour.

Excessive Ingress NGINX restarts indicate that something is wrong. Normally, it should be up and running all the time.

- NginxIngressPodsReportConfigurationHashDrift CE S4
▼ **Ingress NGINX Controller pods report different configuration hashes.**

The `CONTROLLER_NAME` Ingress NGINX Controller pods report different configuration hashes. It can result in inconsistent processing of incoming requests.

To diagnose the issue, follow these steps:

1. Make sure that the pods of `CONTROLLER_NAME` Ingress NGINX Controller are not being updated at the moment. During an update, pods of a single controller may report different config hashes for some time.

```
d8 k -n NAMESPACE get pod -l app=controller,name=CONTROLLER_NAME
```

2. Check the associated metrics `nginx ingress controller_config_hash{namespace="d8-ingress-nginx", controller="CONTROLLER_NAME"}` manually to find the outliers.
3. Check the controller logs for any configuration errors resulting in the configuration drift across the pods:

```
d8 k -n NAMESPACE logs -l app=controller,name=CONTROLLER_NAME -C controller --tail=10000
```

- NginxIngressProfilingIsEnabled CE S4
▼ **Warning: Profiling mode is enabled in the Ingress NGINX Controller "XXX".**

Profiling mode is enabled for the Ingress NGINX Controller “XXX”. This may increase memory consumption, slow down request processing, and the process may run as root. It is recommended to disable profiling unless it is actively needed for debugging or performance analysis. To disable profiling, set `nginxProfilingEnabled: false` in the `ingressnginxcontroller` resource configuration for this controller.

- NginxIngressProtobufExporterHasErrors CE S8
▼ **The Ingress NGINX sidecar container with protobuf_exporter has ERROR_TYPE errors.**

Deckhouse has detected that the Ingress NGINX sidecar container with `protobuf_exporter` has `ERROR_TYPE` errors.

To resolve the issue, check the Ingress controller’s logs:

```
d8 k -n d8-ingress-nginx logs $(d8 k -n d8-ingress-nginx get pods -l
app=controller,name=CONTROLLER_NAME -o wide | grep NODE_NAME | awk '{print $1}') -c
protobuf-exporter
```

- NginxIngressSslExpired CE S4
▼ **Certificate has expired.**

The SSL certificate for `HOST_NAME` in the `NAMESPACE` namespace has expired.

To verify the certificate, run the following command:

```
d8 k -n NAMESPACE get secret SECRET_NAME -o json | jq -r '.data."tls.crt" | @base64d' |
openssl x509 -noout -alias -subject -issuer -dates
```

The site at `https://HOST_NAME` is not accessible.

- NginxIngressSslWillExpire CE S5
▼ **Certificate is expiring soon.**

The SSL certificate for `HOST_NAME` in the `NAMESPACE` namespace will expire in less than two weeks.

To verify the certificate, run the following command:

```
d8 k -n NAMESPACE get secret SECRET_NAME -o json | jq -r '.data."tls.crt" | @base64d' |
openssl x509 -noout -alias -subject -issuer -dates
```

- NginxIngressValidationIsDisabled CE S4
▼ **Warning: Ingress resource validation in the Ingress NGINX Controller is currently disabled.**

Validation is disabled to reduce load on the master nodes, as it requires additional resources. To re-enable validation, remove the annotation `network.deckhouse.io/ingress-nginx-validation-suspended` from the `ingressnginxcontroller` resource.

To find which `IngressNginxController` resources have the annotation, use the following command:

```
d8 k get ingressnginxcontrollers.deckhouse.io -o json | jq -r '.items[] |
select(.metadata.annotations."network.deckhouse.io/ingress-nginx-validation-
suspended" != null) | "\(.metadata.name)">'
```

Модуль istio

- D8IstioActualDataPlaneVersionNotEqualDesired EE S8
▼ **There are pods with Istio data plane version VERSION_NUMBER, but desired version is XXX.**

There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, while the desired version is `XXX`. As a result, the Istio version will be changed after the pod is restarted.

To resolve the issue, use the following cheat sheet:

```
### Namespace-wide configuration
# `istio.io/rev=vXYZ`: Use a specific revision.
# `istio-injection=enabled`: Use the global revision.
d8 k get ns NAMESPACE --show-labels

### Pod-wide configuration
d8 k -n NAMESPACE get pods -l istio.io/rev=DESIRED_VISION
```

- D8IstioActualVersionIsNotInstalled EE S4
▼ **The control plane version for pods with injected sidecars isn't installed.**

There are pods in the `NAMESPACE` namespace with injected sidecars of version `VERSION_NUMBER` (revision `REVISION_NUMBER`), but the corresponding control plane version is not installed. As a result, these pods have lost synchronization with the state in Kubernetes.

To resolve this issue, install the required control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

To identify orphaned pods, run the following command:

```
d8 k -n NAMESPACE get pods -l 'service.istio.io/canonical-name' -o json | jq --arg
revision REVISION_NUMBER '.items[] |
select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
.revision == $revision) | .metadata.name'
```

- D8IstioAdditionalControlplaneDoesntWork CE S4
▼ **Additional control plane isn't working.**

Deckhouse has detected that the additional Istio control plane `ISTIO_REVISION_LABEL` isn't working.

As a result, sidecar injection for pods with `ISTIO_REVISION_LABEL` isn't working as well.

To check the status of the control plane pods, run the following command:

```
d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
```

- **D8IstioDataPlaneVersionMismatch EE S8**

- ▼ **There are pods with data plane version different from the control plane version.**

There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, which is different from the control plane version `DESIRED_VERSION`.

Steps to resolve the issue:

1. Restart affected pods and use the following PromQL query to get a full list:

```
max by (namespace, dataplane_pod)
(d8_istio_dataplane_metadata{full_version="VERSION_NUMBER"})
```

2. Use the automatic Istio data plane upgrade described in the [guide](#).

- **D8IstioDataPlaneWithoutIstioInjectionConfigured EE S4**

- ▼ **Detected pods with Istio sidecars but istio-injection isn't configured.**

There are pods in the `NAMESPACE` namespace with Istio sidecars, but istio-injection isn't configured. As a result, these pods will lose their Istio sidecars after being recreated.

To identify the affected pods, run the following command:

```
d8 k -n NAMESPACE get pods -O json | jq -r --arg revision REVISION_NUMBER '.items[] |
select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
.revision == $revision) | .metadata.name'
```

- **D8IstioDeprecatedIstioVersionInstalled CE**

- ▼ **The installed Istio version has been deprecated.**

Deckhouse has detected that a deprecated Istio version `VERSION_NUMBER` is installed.

Support for this version will be removed in upcoming Deckhouse releases. The higher the alert severity, the greater the probability of support being discontinued.

To learn how to upgrade Istio, refer to the [upgrade guide](#).

- **D8IstioDesiredVersionIsNotInstalled EE S6**

- ▼ **Desired control plane version isn't installed.**

There is a desired Istio control plane version `XXX` (revision `REVISION_NUMBER`) configured for pods in the `NAMESPACE` namespace, but that version isn't installed. As a result, pods can't be recreated in the `NAMESPACE` namespace.

To resolve this issue, install the desired control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

Use the following cheat sheet:

```
### Namespace-wide configuration
# `istio.io/rev=vXYZ`: Use a specific revision.
# `istio-injection=enabled`: Use the global revision.
d8 k get ns NAMESPACE --show-labels

### Pod-wide configuration
d8 k -n NAMESPACE get pods -l istio.io/rev=REVISION_NUMBER
```

- **D8IstioFederationMetadataEndpointDoesntWork EE S6**

- ▼ **Federation metadata endpoint failed.**

The metadata endpoint `ENDPOINT_NAME` for IstioFederation `FEDERATION_NAME` has failed to fetch via the Deckhouse hook.

To reproduce the request to the public endpoint, run the following command:


```
curl ENDPOINT_NAME
```

To reproduce the request to private endpoints (run from the Deckhouse pod), run the following:

```
KEY="$(deckhouse-controller module values istio -o json | jq -r
.internal.remoteAuthnKeypair.priv)"
LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
.global.discovery.clusterUUID)"
REMOTE_CLUSTER_UUID="$(d8 k get istiofederation FEDERATION_NAME -o json | jq -r
.status.metadataCache.public.clusterUUID)"
TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
claim scope=private-federation --ttl 1h)"
curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
```

- D8IstioGlobalControlplaneDoesntWork CE S4

- ▼ **Global control plane isn't working.**

Deckhouse has detected that the global Istio control plane `ISTIO_REVISION_LABEL` isn't working.

As a result, sidecar injection for pods with global revision isn't working as well, and the validating webhook for Istio resources is absent.

To check the status of the control plane pods, run the following command:

```
d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
```

- D8IstioMulticlusterMetadataEndpointDoesntWork EE S6

- ▼ **Multicluster metadata endpoint failed.**

The metadata endpoint `ENDPOINT_NAME` for IstioMulticluster `MULTICLUSTER_NAME` has failed to fetch via the Deckhouse hook.

To reproduce the request to the public endpoint, run the following command:

```
curl ENDPOINT_NAME
```

To reproduce the request to private endpoints (run from the `d8-system/deckhouse` pod), run the following:

```
KEY="$(deckhouse-controller module values istio -o json | jq -r
.internal.remoteAuthnKeypair.priv)"
LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
.global.discovery.clusterUUID)"
REMOTE_CLUSTER_UUID="$(d8 k get istiomulticluster MULTICLUSTER_NAME -o json | jq -r
.status.metadataCache.public.clusterUUID)"
TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
claim scope=private-multicluster --ttl 1h)"
curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
```

- D8IstioMulticlusterRemoteAPIHostDoesntWork EE S6

- ▼ **Multicluster remote API host health check failed.**

The remote API host `API_HOST` for IstioMulticluster `MULTICLUSTER_NAME` has failed the health check performed by the Deckhouse monitoring hook.

To reproduce the request (run from the `d8-system/deckhouse` pod), run the following:

```
TOKEN="$(deckhouse-controller module values istio -o json | jq -r --arg ah API_HOST
'.internal.multiclusters[] | select(.apiHost == $ah) | .apiJWT ')"
curl -H "Authorization: Bearer $TOKEN" https://API_HOST/version
```

- D8IstioOperatorReconcileError CE S5

- ▼ **The istio-operator is unable to reconcile Istio control plane setup.**

Deckhouse has detected an error in the `istio-operator` reconciliation loop.

To investigate the issue, check the operator logs:

```
d8 k -n d8-istio logs -l app=operator,revision=REVISION_NUMBER
```

- D8IstioPodsWithoutIstioSidecar EE S4

▼ **Detected pods without Istio sidecars but with istio-injection configured.**

There is a pod `POD_NAME` in the `NAMESPACE` namespace without Istio sidecars, but with istio-injection configured.

To identify the affected pods, run the following command:

```
d8 k -n NAMESPACE get pods -l '!service.istio.io/canonical-name' -o json | jq -r
'.items[] | select(.metadata.annotations."sidecar.istio.io/inject" != "false")
| .metadata.name'
```

- D8IstioRemoteClusterNotSynced EE S4

▼ **Istio remote cluster `CLUSTER_ID` is not synchronized.**

The Istio control plane instance `INSTANCE_NAME` cannot synchronize with the remote cluster `CLUSTER_ID`.

Possible causes:

- The remote cluster is offline.
- The remote API endpoint is not reachable.
- The remote ServiceAccount token is invalid or expired.
- There is a TLS or certificate issue between the clusters.
- Remote and local Istio versions mismatch.

- D8IstioReservedUIDConflict CE S4

▼ **Application container is running as UID 1337 (reserved by Istio).**

Container `CONTAINER_NAME` in pod `POD_NAME` in namespace `NAMESPACE` is running as UID `1337`.

UID `1337` is reserved by Istio for the `istio-proxy` sidecar container. When application containers run with this UID, their traffic bypasses Istio completely — no routing rules, mTLS, or telemetry will be applied.

To fix the issue, change the `runAsUser` in the container's or pod's `securityContext` to a different UID.

- D8IstioVersionIsIncompatibleWithK8sVersion CE S3

▼ **The installed Istio version is incompatible with the Kubernetes version.**

The installed Istio version `VERSION_NUMBER` may not work properly with the current Kubernetes version `VERSION_NUMBER` because it's not supported officially.

To resolve the issue, upgrade Istio following the guide.

- IstioIrrelevantExternalServiceFound CE S5

▼ **External service found with irrelevant ports specifications.**

A service `NAME` in the `NAMESPACE` namespace has an irrelevant port specification.

The `.spec.ports[]` field isn't applicable for services of the `ExternalName` type. However, Istio renders port listeners for external services as `0.0.0.0:port`, which captures all traffic to the specified port. This can cause problems for services that aren't registered in the Istio registry.

To resolve the issue, remove the `.spec.ports` section from the service configuration. It is safe.

Модуль kube-dns

- KubernetesCoreDNSHasCriticalErrors CE S5

▼ **Critical errors found in CoreDNS.**

Deckhouse has detected at least one critical error in the CoreDNS pod `POD_NAME`.

To resolve the issue, review the container logs:

```
d8 k -n kube-system logs POD_NAME
```

Модуль log-shipper

- D8LogShipperAgentNotScheduledInCluster CE S7
▼ **The log-shipper-agent pods can't be scheduled in the cluster.**

Deckhouse has detected that a number of `log-shipper-agent` pods are not scheduled.

To resolve this issue, do the following:

1. Check the state of the `d8-log-shipper/log-shipper-agent` DaemonSet:

```
d8 k -n d8-log-shipper get daemonsets --selector=app=log-shipper
```

2. Check the state of the `d8-log-shipper/log-shipper-agent` pods:

```
d8 k -n d8-log-shipper get pods --selector=app=log-shipper-agent
```

3. If you know where the DaemonSet should be scheduled, run the following command to identify the problematic nodes:

```
d8 k -n d8-log-shipper get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name == "log-shipper-agent")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type
== "Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.
nodeSelectorTerms[].matchFields[].values[]'
```

- D8LogShipperClusterLogDestinationD8LokiAuthorizationRequired CE S9
▼ **Authorization parameters required for the ClusterLogDestination resource.**

Deckhouse has detected the ClusterLogDestination resource `RESOURCE_NAME` without authorization parameters.

Add the authorization parameters to the ClusterLogDestination resource following the instructions.

- D8LogShipperCollectLogErrors CE S4
▼ **The log-shipper-agent pods can't collect logs to COMPONENT_ID on the NODE_NAME node.**

Deckhouse has detected that the `HOST_NAME` `log-shipper-agent` on the `NODE_NAME` node has failed to collect metrics for more than 10 minutes.

This is caused by the `ERROR_TYPE` errors occurred during the `STAGE_NAME` stage while reading `COMPONENT_TYPE`.

To resolve this, check the pod logs or follow advanced instructions:

```
d8 k -n d8-log-shipper logs HOST_NAME -C vector
```

- D8LogShipperConfigInvalid CE S4
▼ **Vector configuration is invalid in the cluster.**

At least one Vector DaemonSet Pod is reporting an invalid configuration.

To resolve this, check the container logs or follow advanced instructions:

```
d8 k -n d8-log-shipper logs HOST_NAME -C vector-reloader
```

- D8LogShipperDestinationErrors CE S4
▼ **The log-shipper-agent pods can't send logs to COMPONENT_ID on the NODE_NAME node.**

Deckhouse has detected that the `HOST_NAME` `log-shipper-agent` on the `NODE_NAME` node has failed to send a log for more than 10 minutes.

This is caused by the `ERROR_TYPE` errors occurred during the `STAGE_NAME` stage while sending logs to `COMPONENT_TYPE`.

To resolve this, check the pod logs or follow advanced instructions:

```
d8 k -n d8-log-shipper logs HOST_NAME -C vector
```

- D8LogShipperLogsDroppedByRateLimit CE S4
▼ **The log-shipper-agent pods are dropping logs to COMPONENT_ID on the NODE_NAME node.**

Rate-limiting rules have been applied, and the `log-shipper-agent` on the `NODE_NAME` node has been dropping logs for more than 10 minutes.

To resolve this, check the pod logs or follow advanced instructions:

```
d8 k -n d8-log-shipper get pods -O wide | grep NODE_NAME
```

Модуль loki

- LokiDiscardedSamplesWarning CE S4
▼ **Loki is discarding samples during ingestion.**

Loki is discarding samples during ingestion because they fail validation. Samples are being discarded because of `XXX` at a rate of `XXX` samples per second. For more information, refer to the [Loki FAQ](#).

- LokiInsufficientDiskForRetention CE S4
▼ **Not enough disk space to retain logs for 168 hours**

Not enough disk space to retain logs for 168 hours. Current effective retention period is `VALUE` hours.

You need either decrease expected `retentionPeriodHours` or increase `resize Loki PersistentVolumeClaim`

Модуль metallb

- D8MetallbBGPSessionDown EE S4
▼ **MetaLB BGP session is down.**

`JOB_NAME`, `MetaLB CONTAINER_NAME` on `POD_NAME` has BGP session `PEER` down.

Check the logs for details:

```
d8 k -n d8-metallb logs daemonset/speaker -C speaker
```

- D8MetallbConfigNotLoaded EE S4
▼ **The MetaLB configuration hasn't been loaded.**

`JOB_NAME`, `MetaLB CONTAINER_NAME` on `POD_NAME` hasn't been loaded.

To find the cause of the issue, review the controller logs:

```
d8 k -n d8-metallb logs deploy/controller -C controller
```

- D8MetallbConfigStale EE S4
▼ **MetaLB is running on a stale configuration.**

`JOB_NAME`, `MetaLB CONTAINER_NAME` on `POD_NAME` is running on a stale configuration because the latest configuration failed to load.

To find the cause of the issue, review the controller logs:

```
d8 k -n d8-metallb logs deploy/controller -C controller
```

- D8MetallbNotSupportedServiceAnnotationsDetected SE S4
▼ **The annotation 'ANNOTATION_NAME' has been deprecated for the service 'NAME' in the 'NAMESPACE' namespace.**

The annotation `'ANNOTATION_NAME'` has been deprecated for the service `'NAME'` in the `'NAMESPACE'` namespace.

The following service annotations are no longer effective:

- `metallb.universe.tf/ip-allocated-from-pool`: Remove this annotation.
- `metallb.universe.tf/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate `MetalLoadBalancerClass`.

- `metallb.universe.tf/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips`: IP.
- `metallb.universe.tf/allow-shared-ip`: Replace it with `network.deckhouse.io/load-balancer-shared-ip-key`.
- `metallb.io/ip-allocated-from-pool`: Remove this annotation.
- `metallb.io/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate `MetalLoadBalancerClass`.
- `metallb.io/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips`: IP.
- `metallb.io/allow-shared-ip`: Replace it with `network.deckhouse.io/load-balancer-shared-ip-key`.

Please note. Existing LoadBalancer services of Deckhouse have been migrated automatically, but the new ones will not be.

- D8MetallbObsoleteLayer2PoolsAreUsed SE S7
▼ **The metallb module has obsolete layer2 pools configured.**

In ModuleConfig version 2, addressPool 'NAME' of type "layer2" are ignored. They should be removed from the configuration.

- D8MetallbUpdateMCVersionRequired SE S5
▼ **The metallb ModuleConfig settings are outdated.**

D8 MetalLB settings are outdated.

To resolve this issue, increase version in the ModuleConfig `metallb`.

- L2LoadBalancerOrphanServiceFound SE S4
▼ **Orphaned service with an irrelevant L2LoadBalancer name has been found.**

The cluster contains an orphaned service `NAME` in the `NAMESPACE` namespace with an irrelevant L2LoadBalancer name.

To resolve this issue, verify the L2LoadBalancer name specified in the annotation `network.deckhouse.io/l2-load-balancer-name`.

Модуль monitoring-applications

- D8OldPrometheusTargetFormat FE S6
▼ **Services with the prometheus-target label are being used for metric collection.**

Deckhouse has detected that services with the `prometheus-target` label are being used to collect metrics in the cluster.

The label format has been changed. To resolve the issue, replace the `prometheus-target` label with `prometheus.deckhouse.io/target`.

To list all services labeled with `prometheus-target`, run the following command:

```
d8 k get service --all-namespaces --show-labels | grep prometheus-target
```

Модуль monitoring-custom

- CustomPodMonitorFoundInCluster CE S9
▼ **Deckhouse namespace contains PodMonitors not created by Deckhouse.**

There are PodMonitors in the Deckhouse namespace that were not created by Deckhouse.

To resolve the issue, move these PodMonitors to the `user-spec` namespace by removing the `heritage: deckhouse` label.

To list all PodMonitors in the Deckhouse namespace, run the following command:

```
d8 k get podmonitors --all-namespaces -l heritage!=deckhouse
```

For more information on metric collection, refer to the Prometheus module FAQ.

- CustomServiceMonitorFoundInD8Namespace CE S9
▼ **Deckhouse namespace contains ServiceMonitors not created by Deckhouse.**

There are ServiceMonitors in the Deckhouse namespace that were not created by Deckhouse.

To resolve the issue, move these ServiceMonitors to the `user-spec` namespace by removing the `heritage: deckhouse` label.

To list all ServiceMonitors in the Deckhouse namespace, run the following command:

```
d8 k get servicemonitors --all-namespaces -l heritage!=deckhouse
```

For more information on metric collection, refer to the Prometheus module FAQ.

- D8CustomPrometheusRuleFoundInCluster CE S9

▼ **Deckhouse namespace contains PrometheusRules not created by Deckhouse.**

There are PrometheusRules in the Deckhouse namespace that were not created by Deckhouse.

To resolve the issue, replace these PrometheusRules with the CustomPrometheusRules object.

To list all PrometheusRules in the Deckhouse namespace, run the following command:

```
d8 k get prometheusrules --all-namespaces -l heritage!=deckhouse
```

For details on adding alerts and recording rules, refer to the Prometheus module FAQ.

- D8OldPrometheusCustomTargetFormat CE S9

▼ **Services with the prometheus-custom-target label are being used for metric collection.**

Deckhouse has detected that services with the `prometheus-custom-target` label are being used to collect metrics in the cluster.

The label format has been changed. To resolve the issue, replace the `prometheus-custom-target` label with `prometheus.deckhouse.io/custom-target`.

To list all services labeled with `prometheus-custom-target`, run the following command:

```
d8 k get service --all-namespaces --show-labels | grep prometheus-custom-target
```

For more information on metric collection, refer to the Prometheus module FAQ.

- D8ReservedNodeLabelOrTaintFound CE S6

▼ **Node NAME is using a reserved label or taint.**

Deckhouse has detected that node NAME is using one of the following:

- A reserved `metadata.labels` object `node-role.deckhouse.io/`, which doesn't end with `(system|frontend|monitoring|deckhouse module name_)`.
- A reserved `spec.taints` object `dedicated.deckhouse.io`, with a value other than `(system|frontend|monitoring|_deckhouse_module_name_)`.

This alert may fire if the corresponding module is missing or disabled in the cluster. It can be safely ignored if you are preparing nodes to enable a module that is currently disabled.

For instructions on how to resolve this issue, refer to the node allocation guide.

Модуль monitoring-deckhouse

- D8CNIEabledMoreThanOne CE S2

▼ **More than one CNI is enabled in the cluster.**

Deckhouse has detected that multiple CNIs are enabled in the cluster. For the cluster to work correctly, only one CNI must be enabled.

To resolve this issue, disable any unnecessary CNI.

- D8CNIMisconfigured CE S5

▼ **The parameters specified in ModuleConfig of CNI_NAME do not match the ones that are actually being used in the cluster.**

This happened because there were several sources for configuring CNI parameters in the cluster, and ModuleConfig did not have the highest priority previously.

To resolve this issue, the following steps should be taken:

1. Find the ConfigMap `d8-system/desired-cni-moduleconfig` in the cluster, which contains

the actual ModuleConfig settings.

```
d8 k -n d8-system get configmap desired-cni-moduleconfig -O yaml
```

2. Apply this prepared ModuleConfig to the cluster. This will not cause any actual reconfigurations in the cluster and is completely safe.
 3. Once the parameters in “ModuleConfig” match those used in the cluster:
 - this alert will be resolved,
 - and the priority of the sources for configuring CNI will change, and ModuleConfig will become the main source of truth.
 4. After that, you can add the desired parameters to the ModuleConfig CNI and they will be applied immediately.
- D8DeckhouseConfigInvalid CE S5
 - ▼ **Deckhouse configuration is invalid.**

Deckhouse configuration contains errors.

Steps to troubleshoot:

1. Check Deckhouse logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

2. Edit the Deckhouse global configuration:

```
d8 k edit mc global
```

Or edit configuration of a specific module:

```
d8 k edit mc MODULE_NAME
```

- D8DeckhouseCouldNotDeleteModule CE S4
 - ▼ **Deckhouse is unable to delete the MODULE_NAME module.**

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseCouldNotDiscoverModules CE S4
 - ▼ **Deckhouse is unable to discover modules.**

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseCouldNotRunGlobalHook CE S5
 - ▼ **Deckhouse is unable to run the HOOK_NAME global hook.**

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseCouldNotRunModule CE S4
 - ▼ **Deckhouse is unable to start the MODULE_NAME module.**

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseCouldNotRunModuleHook CE S7
 - ▼ **Deckhouse is unable to run the MODULE_NAME/HOOK_NAME module hook.**

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseCustomTargetDown CE S4

Prometheus is unable to scrape custom metrics generated by Deckhouse hooks.

- D8DeckhouseDeprecatedConfigmapManagedByArgoCD CE S4
▼ **Deprecated Deckhouse ConfigMap managed by Argo CD.**

The Deckhouse ConfigMap is no longer used.

To resolve this issue, remove the `d8-system/deckhouse` ConfigMap from Argo CD.

- D8DeckhouseGlobalHookFailsTooOften CE S9
▼ **The HOOK_NAME Deckhouse global hook is crashing too frequently.**

The `HOOK_NAME` hook has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseHasNoAccessToRegistry CE S7
▼ **Deckhouse is unable to connect to the registry.**

Deckhouse can't connect to the registry (typically `registry.deckhouse.io`) to check for a new Docker image. These checks are performed every 15 seconds. Without access to the registry, automatic updates are unavailable.

This alert often indicates that the Deckhouse Pod is experiencing connectivity issues with the Internet.

- D8DeckhouseIsHung CE S4
▼ **Deckhouse is down.**

Deckhouse is probably down, since the `deckhouse_live_ticks` metric in Prometheus has stopped increasing. This metric is expected to increment every 10 seconds.

- D8DeckhouseIsNotOnReleaseChannel CE S9
▼ **Deckhouse isn't subscribed to any regular release channels.**

Deckhouse in this cluster isn't subscribed to any of the regular release channels: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, or `RockSolid`.

To resolve this issue, follow these steps:

1. Check the current release channel used in the cluster:

```
d8 k -n d8-system get deploy deckhouse -o json | jq  
'spec.template.spec.containers[0].image' -r
```

2. Subscribe to one of the regular release channels by adjusting the `deckhouse` module configuration.

- D8DeckhouseModuleHookFailsTooOften CE S9
▼ **The MODULE_NAME/HOOK_NAME Deckhouse hook is crashing too frequently.**

The `HOOK_NAME` hook of the `MODULE_NAME` module has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseModuleUpdatePolicyNotFound CE S5
▼ **Module update policy not found for MODULE_RELEASE.**

The module update policy for `MODULE_RELEASE` is missing.

To resolve this issue, remove the label from the module release using the following command:

```
d8 k label mr MODULE_RELEASE modules.deckhouse.io/update-policy-
```

A new suitable policy will be detected automatically.

- D8DeckhouseModuleValidationError CE S5
▼ **Module configuration failed for module MODULE_NAME.**

Initial config for module MODULE_NAME is not valid.

You can get more details via

```
d8 k get mr -l module=MODULE_NAME
```

Provided error: XXX

- D8DeckhousePodsNotReady CE S4

The Deckhouse Pod is NOT Ready.

- D8DeckhousePodsNotRunning CE S4

The Deckhouse Pod is NOT Running.

- D8DeckhousePodsRestartingTooOften CE S9
▼ **Excessive Deckhouse restarts detected.**

Number of restarts in the last hour: VALUE.

Excessive Deckhouse restarts indicate a potential issue. Normally, Deckhouse should be up and running continuously.

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseQueuesHung CE S7
▼ **The QUEUE_NAME Deckhouse queue is stuck with VALUE pending task(s).**

Deckhouse cannot finish processing of the QUEUE_NAME queue, which currently has VALUE pending task(s).

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseSelfTargetAbsent CE S4

There is no Deckhouse target in Prometheus.

- D8DeckhouseSelfTargetDown CE S4

Prometheus is unable to scrape Deckhouse metrics.

- D8DeckhouseWatchErrorOccurred CE S5
▼ **Possible API server connection error in the client-go informer.**

Deckhouse has detected an error in the client-go informer, possibly due to connection issues with the API server.

Steps to investigate:

1. Check Deckhouse logs for more information by running:

```
d8 k -n d8-system logs deploy/deckhouse | grep error | grep -i watch
```

2. This alert attempts to detect a correlation between the faulty snapshot invalidation and API server connection errors, specifically for the handle-node-template hook in the node-manager module.

To compare the snapshot with the actual node objects for this hook, run the following command:

```
diff -u <(d8 k get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{end}'|sort) <(d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module snapshots node-manager -o json | jq '. "040-node-manager/hooks/handle_node_templates.go"' | jq '.nodes.snapshot[] | .filterResult.Name' -r | sort)
```

- D8HasModuleConfigAllowedToDisable CE S4

▼ The ModuleConfig annotation for allowing module disabling is set.

The ModuleConfig is pending module disabling.

It is recommended that you keep your module configurations clean by removing unnecessary approval annotations.

If you ignore this alert and do not clear the annotation, the module may be accidentally removed from the cluster, potentially leading to irreversible consequences.

To resolve this issue and stop the alert, run the following command:

```
d8 k annotate moduleconfig MODULE_NAME modules.deckhouse.io/allow-disabling-
```

- D8ModuleOutdatedByMajorVersion CE S9

▼ Module MODULE_NAME is running XXX major versions behind.

Module `MODULE_NAME` is running XXX major versions behind on major version releases. Major version updates may contain breaking changes and require immediate attention and careful planning.

To check the module releases status, run the following command:

```
d8 k get mr -l module=MODULE_NAME
```XXXXXX

This major release has from-to version constraints that must be met before applying the update.
To check the update instructions (from-to versions), run the following command:

```bash
d8 k get mr NAME -o jsonpath='{.spec.update.versions[*]}'
```

This will show you the update paths including the from-to versions defined in the release.XXX

To approve the module major release update, run the following command:

```
bash d8 k annotate mr NAME modules.deckhouse.io/approved="true" XXX
```

- D8NodeHasDeprecatedOSVersion CE S4

▼ Nodes with deprecated OS versions detected.

Deckhouse has detected nodes running deprecated OS versions.

Steps to troubleshoot:

1. Get a list of affected nodes by running the following Prometheus query:

```
kube_node_info{os_image=~"Ubuntu 18.04.*|Debian GNU/Linux 10.*|CentOS Linux 7.*"}
```

2. Update the affected nodes to a supported OS version.

- DeckhouseEditionNotFound CE S6

▼ Incorrect Deckhouse Kubernetes Platform edition.

Specify a Deckhouse Kubernetes Platform edition in `MODULE_NAME` ModuleConfig.

```
d8 k patch moduleconfig MODULE_NAME --type merge --patch '{"spec": {"settings": {"license": {"edition": "DKP_EDITION"}}}}'
```

Where `DKP_EDITION` is the edition you want to use: `CE`, `BE`, `EE`, `SE`, or `SE-plus`.

- DeckhouseHighMemoryUsage CE S3

▼ Deckhouse memory usage is too high.

Deckhouse pod `POD_NAME` on node `NODE_NAME` has high memory usage.

Please, if possible, get the dumps for debugging purposes (it will take around 30 seconds):

```
curl -s "http://$(d8 k -n d8-system get pod POD_NAME -O
jsonpath='{.status.hostIP}'):4222/debug/pprof/heap" -o /tmp/heap.pprof
curl -s "http://$(d8 k -n d8-system get pod POD_NAME -O
jsonpath='{.status.hostIP}'):4222/debug/pprof/goroutine" -o /tmp/goroutine.pprof
curl -s "http://$(d8 k -n d8-system get pod POD_NAME -O
jsonpath='{.status.hostIP}'):4222/debug/pprof/profile?seconds=30" -o /tmp/cpu.pprof
```

and send these files (/tmp/heap.pprof, /tmp/cpu.pprof and /tmp/goroutine.pprof) to the support team.

- DeckhouseMigratedModuleNotDownloaded CE S6
▼ **Migrated module MODULE_NAME has not been downloaded yet.**

Module MODULE_NAME specified in DeckhouseRelease was moved to an external source and is available in a ModuleSource registry, but it has not been downloaded yet (no deployed ModuleRelease exists). DeckhouseRelease will be locked until the module is downloaded.

This alert is usually temporary while the module source controller downloads the module from the registry. If the alert persists, this may indicate:

- The module's update policy or ModuleSource is misconfigured, preventing the module from being downloaded.
- A network issue is preventing the module from being downloaded.

To investigate the issue, do the following:

- Check the ModuleRelease objects for the module:

```
d8 k get mr -l module=MODULE_NAME
```

- Check the configured ModuleSources:

```
d8 k get modulesources
```

- Check the deckhouse logs for more details:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- DeckhouseMigratedModuleNotFound CE S6
▼ **Migrated module MODULE_NAME was not found in any ModuleSource registry.**

Module MODULE_NAME specified in DeckhouseRelease was moved to an external source and was not found in any available ModuleSource registry. DeckhouseRelease will be locked until the module becomes available in one of the configured ModuleSource registries.

This may indicate:

- A ModuleSource registry is misconfigured.
- The module has not been published to the registry yet.
- A network issue is preventing access to the registry.

To investigate the issue, do the following:

- Check the configured ModuleSources:

```
d8 k get modulesources
```

- List the available module sources:

```
d8 k -n d8-system exec -it deployment/deckhouse -C deckhouse -- deckhouse-controller
registry get sources
```

- List the modules available in a specific source:

```
d8 k -n d8-system exec -it deployment/deckhouse -C deckhouse -- deckhouse-controller
registry get modules source_name
```

- Check the deckhouse logs for more details:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- DeckhouseModuleUpdatingFailedBrokenSequence CE S4
▼ **Module update has failed. Updating sequence is broken.**

The 'MODULE_NAME' Module update has failed.

Current version: XXX. Desired version: VERSION_NUMBER.

Attempt to download interim releases failed.

Possible reasons:

- The necessary versions of the Module image is not available in the registry.

To resolve this issue, ensure that all minor versions of the 'MODULE_NAME' Module image between XXX and VERSION_NUMBER is available in the registry 'XXX'.

- DeckhouseModuleUpdatingFailedModuleIsNotValid CE S4
▼ **Module update has failed. Module is not valid.**

The 'MODULE_NAME' Module update has failed.

Desired version: VERSION_NUMBER.

Possible reasons:

- The Module image is corrupted.

To resolve this issue, ensure that VERSION_NUMBER image in the registry 'XXX' is valid .

To get the exact error, check Deckhouse logs: `d8 k -n d8-system logs deploy/deckhouse | grep "failed to download module"`

- DeckhouseReleaseDisruptionApprovalRequired CE S4
▼ **Deckhouse release disruption approval required.**

The new Deckhouse release includes a disruptive update that requires manual approval.

To check the details, run the following command:

```
d8 k describe DeckhouseRelease NAME
```

To approve the disruptive update, run the following command:

```
d8 k annotate DeckhouseRelease NAME release.deckhouse.io/disruption-approved=true
```

- DeckhouseReleasesBlocked CE S5
▼ **Deckhouse release requirements haven't been met.**

The requirements for the Deckhouse release haven't been met.

To check the details, run the following command:

```
d8 k describe DeckhouseRelease NAME
```

- DeckhouseReleasesWaitingManualApproval CE S3
▼ **A new Deckhouse release is awaiting manual approval.**

A new Deckhouse release is available but requires manual approval before it can be applied.

To approve the release, run the following command:

```
d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
```

- DeckhouseReleasesWaitingManualApproval CE S6
▼ **A new Deckhouse release is awaiting manual approval.**

A new Deckhouse release is available but requires manual approval before it can be applied.

To approve the release, run the following command:

```
d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
```

- DeckhouseReleasesWaitingManualApproval CE S9
▼ **A new Deckhouse release is awaiting manual approval.**

A new Deckhouse release is available but requires manual approval before it can be applied.

To approve the release, run the following command:

```
d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
```

- DeckhouseReleaseNotificationNotSent CE S4
▼ **Deckhouse release notification webhook hasn't been sent.**

The Deckhouse release notification webhook failed to send.

To check the notification webhook address, run the following command:

```
d8 k get mc deckhouse -O yaml
```

- DeckhouseUpdating CE S4

Deckhouse is being updated to XXX.

- DeckhouseUpdatingFailed CE S4
▼ **Deckhouse update has failed.**

The Deckhouse update has failed.

Possible reasons:

- The next minor or patch version of the Deckhouse image is not available in the registry.
- The Deckhouse image is corrupted.

Current version: VERSION_NUMBER.

To resolve this issue, ensure that the next version of the Deckhouse image is available in the registry.

- ModuleAtConflict CE S4
▼ **Conflict detected for module MODULE_NAME.**

Deckhouse has detected conflicting sources for the MODULE_NAME module.

To resolve this issue, specify the correct source in the module configuration.

- ModuleConfigObsoleteVersion CE S4
▼ **ModuleConfig NAME is outdated.**

Deckhouse has detected that ModuleConfig NAME is outdated.

To resolve this issue, update ModuleConfig NAME to the latest version.

- ModuleHasDeprecatedUpdatePolicy CE S4
▼ **Module MODULE_NAME is matched by the deprecated update policy UPDATE_POLICY_NAME.**

Deckhouse has detected that the module MODULE_NAME is using the deprecated update policy UPDATE_POLICY_NAME. The v1alpha1 policy has a selector that no longer functions.

To specify the update policy in the module configuration, run the following command:

```
d8 k patch moduleconfig MODULE_NAME --type='json' -p='[{"op": "add", "path":  
"/spec/updatePolicy", "value": "UPDATE_POLICY_NAME"}]'
```

After resolving all alerts related to the update policy UPDATE_POLICY_NAME, clear the selector by running the following command:

```
d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io UPDATE_POLICY_NAME --type='json' -  
p='[{"op": "replace", "path":  
"/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}]'
```

- ModulesDeprecated CE S9
▼ **Module MODULE_NAME is deprecated.**

The module MODULE_NAME is deprecated and will not receive new updates. The module may be

removed in the future, so please disable it in a controlled manner.

- ModuleIsInMaintenanceMode CE S6
▼ **Module MODULE_NAME is running in maintenance mode.**

Module MODULE_NAME is running in maintenance mode. In this mode, its state is not reconciled, which prevents configuration or hook changes from being applied automatically.

To switch the module back to normal mode, edit the corresponding ModuleConfig resource with the following command:

```
d8 k patch moduleconfig MODULE_NAME --type=json -p='[{"op": "remove", "path":
"/spec/maintenance"}]'
```

- ModuleReleasesBlockedByRequirements CE S6
▼ **A new release for module MODULE_NAME has been blocked due to unmet requirements.**

A new release for module MODULE_NAME has been blocked because it doesn't meet the required conditions.

To check the requirements, run the following command:

```
d8 k get mr NAME -O json | jq .spec.requirements
```

- ModuleReleasesOutdated CE S5
▼ **Module MODULE_NAME is XXX or more minor versions behind the latest available release.**

Module MODULE_NAME is running XXX or more minor versions behind the latest available release. This module is significantly outdated and may have compatibility issues.

To approve the module release update, run the following command:

```
d8 k annotate mr NAME modules.deckhouse.io/approved="true"
```

- ModuleReleasesOutdated CE S7
▼ **Module MODULE_NAME is 2 minor versions behind the latest available release.**

Module MODULE_NAME is running 2 minor versions behind the latest available release. Updating is recommended to maintain compatibility and access to latest features.

To approve the module release update, run the following command:

```
d8 k annotate mr NAME modules.deckhouse.io/approved="true"
```

- ModuleReleasesOutdated CE S9
▼ **Module MODULE_NAME is 1 minor version behind the latest available release.**

Module MODULE_NAME is running 1 minor version behind the latest available release. Consider updating to maintain compatibility and security.

To approve the module release update, run the following command:

```
d8 k annotate mr NAME modules.deckhouse.io/approved="true"
```

- ModuleReleasesWaitingManualApproval CE S6
▼ **A new release for module MODULE_NAME is awaiting manual approval.**

A new release for module MODULE_NAME is available but requires manual approval before it can be applied.

To approve the module release, run the following command:

```
d8 k annotate mr NAME modules.deckhouse.io/approved="true"
```

Модуль monitoring-kubernetes

- CPUStealHigh CE S4
▼ **CPU steal on node NODE_NAME is too high.**

The CPU steal has been excessively high on node `NODE_NAME` over the past 30 minutes.

Another component, such as a neighboring virtual machine, may be consuming the node's resources. This may be the result of "overselling" the hypervisor, meaning the hypervisor is hosting more virtual machines than it can handle.

- DeadMansSwitch CE S4
▼ **Alerting dead man's switch.**

This is a dead man's switch meant to ensure that the entire Alerting pipeline is functional.

- DeploymentGenerationMismatch CE S4
▼ **Deployment is outdated.**

The observed deployment generation doesn't match the expected one for Deployment `NAMESPACE/DEPLOYMENT_NAME`.

- FdExhaustionClose CE S3
▼ **File descriptors for JOB_NAME: INSTANCE_NAME are almost exhausted.**

The instance `JOB_NAME: INSTANCE_NAME` is expected to exhaust its available file/socket descriptors within the next hour.

- FdExhaustionClose CE S3
▼ **File descriptors for JOB_NAME: NAMESPACE/POD_NAME are almost exhausted.**

The instance `JOB_NAME: NAMESPACE/POD_NAME` is expected to exhaust its available file/socket descriptors within the next hour.

- FdExhaustionClose CE S4
▼ **File descriptors for JOB_NAME: INSTANCE_NAME are exhausting soon.**

The instance `JOB_NAME: INSTANCE_NAME` is expected to exhaust its available file/socket descriptors within the next 4 hours.

- FdExhaustionClose CE S4
▼ **File descriptors for JOB_NAME: NAMESPACE/POD_NAME are exhausting soon.**

The instance `JOB_NAME: NAMESPACE/POD_NAME` is expected to exhaust its available file/socket descriptors within the next 4 hours.

- HelmReleasesHasResourcesWithDeprecatedVersions CE S5
▼ **At least one Helm release contains resources with deprecated apiVersion, which will be removed in Kubernetes version VERSION_NUMBER.**

To list all affected resources, run the following Prometheus query:

```
max by (helm_release_namespace, helm_release_name, helm_version,
resource_namespace, resource_name, api_version, kind, k8s_version)
(resource_versions_compatibility) == 1
```

For more details on upgrading deprecated resources, refer to the Kubernetes deprecation guide available at <https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX>.

Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- HelmReleasesHasResourcesWithUnsupportedVersions CE S4
▼ **At least one Helm release contains resources with unsupported apiVersion for Kubernetes version VERSION_NUMBER.**

To list all affected resources, run the following Prometheus query:

```
max by (helm_release_namespace, helm_release_name, helm_version,
resource_namespace, resource_name, api_version, kind, k8s_version)
(resource_versions_compatibility) == 2
```

For more details on migrating deprecated resources, refer to the Kubernetes deprecation guide available at <https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX>.

Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- K8SKubeletDown CE S3
▼ **Multiple kubelets couldn't be scraped.**

Prometheus failed to scrape VALUE% of kubelets.
- K8SKubeletDown CE S4
▼ **Several kubelets couldn't be scraped.**

Prometheus failed to scrape VALUE% of kubelets.
- K8SKubeletTooManyPods CE S7
▼ **The kubelet on node NODE_NAME is approaching the pod limit.**

The kubelet on node NODE_NAME is running VALUE pods, which is close to the limit of XXX.
- K8SManyNodesNotReady CE S3
▼ **Too many nodes are not ready.**

VALUE% of Kubernetes nodes are not ready.
- K8SNodeNotReady CE S3
▼ **The status of node NODE_NAME is NotReady.**

The kubelet on node NODE_NAME has either failed to check in with the API server or has set itself to `NotReady` for more than 10 minutes.
- KubeletImageFSBytesUsage CE S5

No free space remaining on imagefs on node NODE_NAME at mountpoint MOUNTPOINT.
- KubeletImageFSBytesUsage CE S6
▼ **Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.**

Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is currently in progress.

Hard eviction threshold: XXX%

Current usage: VALUE%
- KubeletImageFSBytesUsage CE S7
▼ **Approaching hard eviction threshold of imagefs on node NODE_NAME at mountpoint MOUNTPOINT.**

The imagefs usage on node NODE_NAME at mountpoint MOUNTPOINT is nearing the hard eviction threshold.

Hard eviction threshold: XXX%

Current usage: VALUE%
- KubeletImageFSBytesUsage CE S9
▼ **Soft eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.**

Soft eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is currently in progress.

Soft eviction threshold: XXX%

Current usage: VALUE%
- KubeletImageFSInodesUsage CE S5

No free inodes remaining on imagefs on node NODE_NAME at mountpoint MOUNTPOINT.
- KubeletImageFSInodesUsage CE S6
▼ **Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.**

Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is currently in progress.

Hard eviction threshold: XXX%

Current usage: VALUE%
- KubeletImageFSInodesUsage CE S7
▼ **Approaching hard eviction threshold of imagefs on node NODE_NAME at mountpoint MOUNTPOINT.**

The imagefs usage on node NODE_NAME at mountpoint MOUNTPOINT is nearing the hard eviction

The images usage on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is nearing the hard eviction threshold.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletImageFSInodesUsage CE S9

▼ **Soft eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is in progress.**

Soft eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is currently in progress.

Soft eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S5

No free space remaining on nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT`.

- KubeletNodeFSBytesUsage CE S6

▼ **Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is in progress.**

Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is currently in progress.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S7

▼ **Approaching hard eviction threshold of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT`.**

The nodefs usage on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is nearing the hard eviction threshold.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S9

▼ **Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is in progress.**

Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is currently in progress.

Soft eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S5

No free inodes remaining on nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT`.

- KubeletNodeFSInodesUsage CE S6

▼ **Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is in progress.**

Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is currently in progress.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S7

▼ **Approaching hard eviction threshold of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT`.**

The nodefs usage on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is nearing the hard eviction threshold.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S9

▼ **Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is in progress.**

Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPPOINT` is currently in progress.

Soft eviction threshold: XXX%

Current usage: VALUE%

- **KubernetesDnsTargetDown CE S5**
▼ **Kube-dns or CoreDNS are not being monitored.**

Prometheus is unable to collect metrics from `kube-dns`, which makes its status unknown.

Steps to troubleshoot:

1. Check the deployment details:

```
d8 k -n kube-system describe deployment -l k8s-app=kube-dns
```

2. Check the pod details:

```
d8 k -n kube-system describe pod -l k8s-app=kube-dns
```

- **KubeStateMetricsDown CE S3**
▼ **Kube-state-metrics isn't working in the cluster.**

Deckhouse has detected that no metrics about cluster resources have been available for the past 5 minutes. As a result, most alerts and monitoring panels aren't working.

Steps to troubleshoot:

1. Check the `kube-state-metrics` pods:

```
d8 k -n d8-monitoring describe pod -l app=kube-state-metrics
```

2. Check the deployment logs:

```
d8 k -n d8-monitoring describe deploy kube-state-metrics
```

- **LoadBalancerServiceWithoutExternalIP CE S4**
▼ **A LoadBalancer has not been created.**

One or more services of the `LoadBalancer` type have not received an external address.

To list affected services, run the following command:

```
d8 k get svc -Ao json | jq -r '.items[] | select(.spec.type == "LoadBalancer") | select(.status.loadBalancer.ingress[0].ip == null) | "namespace: \n(.metadata.namespace), name: \n(.metadata.name), ip: \n(.status.loadBalancer.ingress[0].ip)"'
```

Steps to troubleshoot:

- Check the `cloud-controller-manager` logs in the `d8-cloud-provider-*` namespace.
- If you are using a bare-metal cluster with the `metallb` module enabled, ensure the address pool has not been exhausted.

- **NodeConntrackTableFull CE S3**
▼ **The conntrack table on node NODE_NAME is full.**

As a result, no new connections created or accepted on the node. Keeping the `conntrack` table at full capacity may lead to erratic software behavior.

To identify the source of excessive `conntrack` entries, use Okmeter or Grafana dashboards.

- **NodeConntrackTableFull CE S4**
▼ **The conntrack table on node NODE_NAME is approaching the size limit.**

The `conntrack` table on node `NODE_NAME` is currently at VALUE% of its maximum capacity.

This is acceptable as long as the `conntrack` table remains 70-80% full. However, if it reaches full capacity, new connections may fail, causing network disruptions and erratic software behavior.

To identify the source of excessive `conntrack` entries, use Okmeter or Grafana dashboards.

- NodeExporterDown CE S3
▼ **Prometheus couldn't scrape a node-exporter.**

Prometheus has been unable to scrape a `node-exporter` for more than 10 minutes, or `node-exporters` have disappeared from service discovery.

- NodeFilesystemIsRO CE S4
▼ **Node file system is read-only.**

The file system on the node has been switched to read-only mode.

To investigate the cause, check the node logs.

- NodeSunreclaimBytesUsageHigh CE S4
▼ **Node `NODE_NAME` has high kernel memory usage.**

The node `NODE_NAME` has a potential kernel memory leak. One known issue could be causing this problem.

Steps to troubleshoot:

1. Check the `cgroupDriver` setting on node `NODE_NAME`:

```
cat /var/lib/kubelet/config.yaml | grep 'cgroupDriver: systemd'
```

2. If `cgroupDriver` is set to `systemd`, a reboot is required to switch back to the `cgroupfs` driver. In this case, drain and reboot the node.

For further details, refer to the [issue](#).

- NodeSystemExporterDoesNotExistsForNode CE S4
▼ **Some system exporters aren't functioning correctly on node `NODE_NAME`.**

To resolve the issue, follow these steps:

1. Find the `node-exporter` Pod running on the affected node:

```
d8 k -n d8-monitoring get pod -l app=node-exporter -o json | jq -r ".items[] | select(.spec.nodeName==\"NODE_NAME\") | .metadata.name"
```

2. Describe the `node-exporter` Pod:

```
d8 k -n d8-monitoring describe pod POD_NAME
```

3. Verify that the kubelet is running on node `NODE_NAME`.

- NodeTCPMemoryExhaust CE S6
▼ **Node `NODE_NAME` has high TCP stack memory usage.**

The TCP stack on node `NODE_NAME` is experiencing high memory pressure. This may be caused by applications with intensive TCP networking usage.

Steps to troubleshoot:

1. Identify applications consuming excessive TCP memory.
2. Adjust TCP memory configuration if needed.
3. Investigate network traffic sources.

- NodeUnschedulable CE S8
▼ **Node `NODE_NAME` is cordon-protected, preventing new pods from being scheduled.**

The node `NODE_NAME` is in a cordon-protected state, meaning no new pods can be scheduled onto it.

Someone may have executed one of the following commands on this node:

- To cordon the node:

```
d8 k cordon NODE_NAME
```

- To drain the node (if draining has been running for more than 20 minutes):

```
d8 k drain NODE_NAME
```

The was likely caused by the scheduled maintenance of that node.

- PodOOMKilled CE S4
▼ **Pod was OOMKilled.**

The container was killed by the Linux OOM Killer.

- Namespace: NAMESPACE
- Pod: POD_NAME
- Container: XXX

Check memory limits and actual memory usage.

- PodOOMKilled CE S4
▼ **Pod was OOMKilled.**

The container was killed by the Linux OOM Killer.

- Namespace: NAMESPACE
- Pod: POD_NAME
- Container: XXX

Check memory limits and actual memory usage.

- PodStatusIsIncorrect CE S6
▼ **Incorrect state of Pod NAMESPACE/POD_NAME running on node NODE_NAME.**

The Pod NAMESPACE/POD_NAME running on node NODE_NAME is listed as NotReady while all the Pod's containers are Ready.

This could have been caused by the known Kubernetes bug.

Steps to troubleshoot:

1. Identify all pods with this state:

```
d8 k get pod -o json --all-namespaces | jq '.items[] | select(.status.phase == "Running") | select(.status.conditions[] | select(.type == "ContainersReady" and .status == "True")) | select(.status.conditions[] | select(.type == "Ready" and .status == "False")) | "\(.spec.nodeName)/\(.metadata.namespace)/\(.metadata.name) "'
```

2. Identify all affected nodes:

```
d8 k get pod -o json --all-namespaces | jq '.items[] | select(.status.phase == "Running") | select(.status.conditions[] | select(.type == "ContainersReady" and .status == "True")) | select(.status.conditions[] | select(.type == "Ready" and .status == "False")) | .spec.nodeName' -r | sort | uniq -c
```

3. Restart kubelet on each node:

```
systemctl restart kubelet
```

- StorageClassCloudManual CE S6
▼ **Manually deployed StorageClass NAME found in the cluster.**

A StorageClass using a cloud-provider provisioner shouldn't be deployed manually. Such StorageClasses are managed by the cloud-provider module.

Instead of the manual deployment, modify the cloud-provider module configuration as needed.

- StorageClassDefaultDuplicate CE S6
▼ **Multiple default StorageClasses found in the cluster.**

Deckhouse has detected that more than one StorageClass in the cluster is annotated as default.

This may have been caused by a manually deployed StorageClass that is overlapping with the default storage configuration provided by the cloud-provider module.

- UnsupportedContainerRuntimeVersion CE
▼ **Unsupported version of CRI Containerd X.XX.XXX installed for Kubernetes version VERSION_NUMBER.**

Deckhouse has detected that the node `NODE_NAME` is running an unsupported version of CRI Containerd `X.XX.XXX`.

Supported CRI versions for Kubernetes `VERSION_NUMBER`:

- Containerd 1.4.
- Containerd 1.5.
- Containerd 1.6.
- Containerd 1.7.

Модуль `monitoring-kubernetes-control-plane`

- K8SApiserverDown CE S3
▼ **API servers can't be reached.**

No API servers are reachable, or they have all disappeared from service discovery.

- K8sCertificateExpiration CE S5
▼ **Kubernetes has API clients with soon-to-expire certificates.**

Some clients are connecting to `COMPONENT_NAME` with certificates that will expire in less than a day on node `COMPONENT_NAME`.

To check control plane certificates, use `kubeadm`:

1. Install `kubeadm` using the following command:

```
apt install kubeadm=1.24.*
```

2. Check certificates:

```
kubeadm alpha certs check-expiration
```

To check kubelet certificates, do the following on each node:

1. Check kubelet configuration:

```
ps aux \
| grep "/usr/bin/kubelet" \
| grep -o -e "--kubeconfig=\S*" \
| cut -f2 -d"=" \
| xargs cat
```

2. Locate the `client-certificate` or `client-certificate-data` field.
3. Check certificate expiration using `OpenSSL`.

Note that there are no tools to find other stale `kubeconfig` files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8sCertificateExpiration CE S6
▼ **Kubernetes has API clients with soon-to-expire certificates.**

Some clients are connecting to `COMPONENT_NAME` with certificates that will expire in less than 7 days on node `NODE_NAME`.

To check control plane certificates, use `kubeadm`:

1. Install `kubeadm` using the following command:

```
apt install kubeadm=1.24.*
```

2. Check certificates:

```
kubeadm alpha certs check-expiration
```

To check kubelet certificates, do the following on each node:

1. Check kubelet configuration:

```
ps aux \
| grep "/usr/bin/kubelet" \
| grep -o -e "--kubeconfig=\S*" \
| cut -f2 -d"=" \
| xargs cat
```

2. Locate the `client-certificate` or `client-certificate-data` field.
3. Check certificate expiration using OpenSSL.

Note that there are no tools to find other stale kubeconfig files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8SControllerManagerTargetDown CE S3
▼ **Controller manager is down.**

There is no running kube-controller-manager. As a result, deployments and replication controllers are not progressing.

- K8SSchedulerTargetDown CE S3
▼ **Scheduler is down.**

The Kubernetes scheduler is not running. As a result, new pods are not being assigned to nodes.

- KubeEtcdHighFsyncDurations CE S7
▼ **Syncing (fsync) WAL files to disk is slow.**

In the last 15 minutes, the 99th percentile of the fsync duration for WAL files exceeded 0.5 seconds: VALUE.

Possible causes:

- High disk latency where etcd data is stored.
- High CPU usage on the node.
- KubeEtcdHighNumberOfLeaderChanges CE S5
▼ **The etcd cluster is re-electing the leader too frequently.**

There have been VALUE leader re-elections for the etcd cluster member running on node `NODE_NAME` in the last 10 minutes.

Possible causes:

- High disk latency where etcd data is stored.
- High CPU usage on the node.
- Degradation of network connectivity between cluster members in the multi-master mode.
- KubeEtcdInsufficientMembers CE S4
▼ **Insufficient members in the etcd cluster.**

The etcd cluster has too few members, increasing the risk of failure if another member becomes unavailable. To resolve this issue, check the status of etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd`

- KubeEtcdNoLeader CE S4
▼ **The etcd cluster member running on node `NODE_NAME` has lost the leader.**

To resolve this issue, check the status of the etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd | grep NODE_NAME`

- KubeEtcdTargetAbsent CE S5
▼ **There is no etcd target in Prometheus.**

Steps to troubleshoot:

1. Check the status of the etcd Pods:

```
d8 k -n kube-system get pod -l component=etcd
```

2. Review Prometheus logs:

```
d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -C prometheus
```

- KubeEtcdTargetDown CE S5
▼ **Prometheus is unable to scrape etcd metrics.**

Steps to troubleshoot:

1. Check the status of the etcd Pods:

```
d8 k -n kube-system get pod -l component=etcd
```

2. Review Prometheus logs:

```
d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -C prometheus
```

Модуль monitoring-ping

- NodePingPacketLoss CE S4
▼ **Ping loss exceeds 5%.**

ICMP packet loss to node `NODE_NAME` has exceeded 5%.

Модуль node-local-dns

- D8NodeLocalDNSCacheDenialEvictionsHigh BE S6
▼ **node-local-dns denial cache is being frequently evicted.**

The `denial` (NXDOMAIN) cache in node-local-dns is evicting entries frequently.

This might indicate that there are Services that are being recreated too often in cluster. So, denial cache size is not enough (`denial 65536` is currently configured).

Affected node: `NODE_NAME`

- D8NodeLocalDNSCacheSuccessEvictionsHigh BE S6
▼ **node-local-dns success cache is being frequently evicted.**

The `success` cache in node-local-dns is evicting entries frequently (>100 in 5 minutes).

This could mean there are too many different DNS records application have to resolve (to many Services?). So, node-local-dns cache capacity is too small (`success 120000` is currently configured), or traffic patterns changed.

Affected node: `NODE_NAME`

- D8NodeLocalDNSForwardMaxConcurrentRejects BE S5
▼ **The forward plugin is rejecting DNS queries due to max concurrency limit.**

More than 20 queries were rejected within the last 5 minutes on node `NODE_NAME` because the max concurrent query limit was reached.

- D8NodeLocalDNSKubeforwardRequestLatencyP95High BE S4
▼ **High DNS request latency in kubeforward (p95 > 300ms under load).**

The 95th percentile DNS request latency for A-records through kubeforward exceeded 300 milliseconds over the last 5 minutes while handling more than 0.1 QPS.

This may indicate slow upstreams, network issues, or DNS overload on this node.

Affected node: `NODE_NAME`

For more detailed diagnostics, inspect `Top 5 upstreams by slow kubeforward requests` in Grafana.

If the alert fires only on one node, run the checks below from node `NODE_NAME` (for example, from a debug pod on this node). If the alert fires on many nodes, treat it as a potential systemic issue first.

Get kubeforward upstream IPs from a master host:

```
d8 k -n kube-system get po -l k8s-app=kube-dns -o jsonpath='{.items[*].status.podIP}'
```

Then run latency checks on the affected node `NODE_NAME` (or from a debug pod on that node):

```
UPSTREAMS="paste kube-dns pod IPs here"
for ip in $UPSTREAMS; do
  echo "== upstream ${ip} =="
  for i in $(seq 1 20); do
    dig @"$ip" -p 5353 kubernetes.default.svc.cluster.local A +tries=1 +time=1 +stats | grep
    "Query time"
  done
done
```

To identify slow requests, enable module logs (`enableLogs`) and inspect slow query messages:

```
pod=$(kubectl -n kube-system get po -l k8s-app=node-local-dns --field-selector
spec.nodeName=NODE_NAME -o jsonpath='{.items[0].metadata.name}')
d8 k -n kube-system logs "$pod" | grep '\[kubeforward\] slow query'
```

Example log line: `[kubeforward] slow query A kubernetes.default.svc.cluster.local. took 425ms (rcode=NOERROR, upstream=10.96.0.10:53)`

Interpretation:

- If most upstreams are slow and many nodes are affected, suspect systemic network/CoreDNS overload.
- If one upstream is slow or only one node is affected, suspect a localized issue.
- D8NodeLocalDNSUpstreamHealthcheckFailed BE S5
▼ **Too many upstream health check failures in forward plugin.**

The health check system of the `forward` plugin failed more than 10 times within 5 minutes on node `NODE_NAME`. As a result, all upstreams were considered unhealthy.

Модуль node-manager

- CapsInstanceUnavailable CE S8
▼ **There are unavailable instances in the MACHINE_DEPLOYMENT_NAME MachineDeployment.**

The MachineDeployment `MACHINE_DEPLOYMENT_NAME` has **VALUE** unavailable instances.

Check the status of the instances in the cluster with the following command:

```
d8 k get instance -l node.deckhouse.io/group=MACHINE_DEPLOYMENT_NAME
```

- ClusterHasOrphanedDisks CE S6
▼ **Cloud data discoverer found orphaned disks in the cloud.**

The cloud data discoverer has found disks in the cloud that do not have a corresponding PersistentVolume in the cluster.

You can safely delete these disks manually from your cloud provider:

ID: ID, Name: NAME

- D8BashibleApiserverLocked CE S6
▼ **Bashible-apiserver has been locked for too long.**

`Bashible-apiserver` has been locked for an extended period.

To resolve the issue, check if the `bashible-apiserver` Pods are up-to-date and running:

```
d8 k -n d8-cloud-instance-manager get pods -l app=bashible-apiserver
```

- D8CloudDataDiscovererCloudRequestError CE S6
▼ **Cloud data discoverer cannot get data from the cloud.**

Refer to the cloud data discoverer's logs for details:

```
d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
```

- D8CloudDataDiscovererSaveError CE S6
▼ **Cloud data discoverer cannot save data to a Kubernetes resource.**

Refer to the cloud data discoverer's logs for details:

```
d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
```

- D8ClusterAutoscalerManagerPodsNotReady CE S8
▼ **The POD_NAME Pod is NOT Ready.**

To check the Pod's status, run the following command:

```
d8 k -n NAMESPACE logs POD_NAME
```

- D8ClusterAutoscalerPodsNotRunning CE S8
▼ **The cluster-autoscaler Pod is NOT Running.**

The POD_NAME Pod is STATUS.

To check the Pod's status, run the following command:

```
d8 k -n NAMESPACE get pods POD_NAME -O json | jq .status
```

- D8ClusterAutoscalerPodsRestartingTooOften CE S9
▼ **Too many cluster-autoscaler restarts detected.**

The cluster-autoscaler has restarted VALUE times in the past hour.

Frequent restarts may indicate a problem. The cluster-autoscaler is expected to run continuously without interruption.

Check the logs for details:

```
d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -C cluster-autoscaler
```

- D8ClusterAutoscalerTargetAbsent CE S8
▼ **Cluster-autoscaler target is missing in Prometheus.**

The cluster-autoscaler automatically scales nodes in the cluster. If it's unavailable, it will be impossible to add new nodes when there's not enough resources for scheduling Pods. It may also lead to unnecessary cloud costs due to unused but still provisioned cloud instances.

To resolve the issue, follow these steps:

1. Check availability and status of cluster-autoscaler Pods:

```
d8 k -n d8-cloud-instance-manager get pods -l app=cluster-autoscaler
```

2. Verify that the cluster-autoscaler Deployment exists:

```
d8 k -n d8-cloud-instance-manager get deploy cluster-autoscaler
```

3. Check the Deployment's status:

```
d8 k -n d8-cloud-instance-manager describe deploy cluster-autoscaler
```

- D8ClusterAutoscalerTargetDown CE S8

Prometheus is unable to scrape cluster-autoscaler's metrics.

- D8ClusterAutoscalerTooManyErrors CE S8
▼ **Cluster-autoscaler is issuing too many errors.**

The cluster-autoscaler encountered multiple errors from the cloud provider when attempting to scale the cluster.

Check the logs for details:

```
d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -C cluster-autoscaler
```

- D8MachineControllerManagerPodsNotReady CE S8

The POD NAME Pod is NOT Ready.

- D8MachineControllerManagerPodsNotRunning CE S8
▼ **The machine-controller-manager Pod is NOT Running.**

The POD_NAME Pod is STATUS.

To check the Pod's status, run the following command:

```
d8 k -n NAMESPACE get pods POD_NAME -o json | jq .status
```

- D8MachineControllerManagerPodsRestartingTooOften CE S9
▼ **Too many machine-controller-manager restarts detected.**

The machine-controller-manager has restarted VALUE times in the past hour.

Frequent restarts may indicate a problem. The machine-controller-manager is expected to run continuously without interruption.

Check the logs for details:

```
d8 k -n d8-cloud-instance-manager logs -f -l app=machine-controller-manager -C controller
```

- D8MachineControllerManagerTargetAbsent CE S8
▼ **Machine-controller-manager target is missing in Prometheus.**

Machine-controller-manager controls ephemeral nodes in the cluster. If it becomes unavailable, it will be impossible to create or delete nodes in the cluster.

To resolve the issue, follow these steps:

1. Check availability and status of machine-controller-manager Pods:

```
d8 k -n d8-cloud-instance-manager get pods -l app=machine-controller-manager
```

2. Verify availability of the machine-controller-manager Deployment:

```
d8 k -n d8-cloud-instance-manager get deployment machine-controller-manager
```

3. Check the Deployment's status:

```
d8 k -n d8-cloud-instance-manager describe deployment machine-controller-manager
```

- D8MachineControllerManagerTargetDown CE S8

Prometheus is unable to scrape the machine-controller-manager's metrics.

- D8NeedMigrateStateToOpenTofu CE S4
▼ **Terraform state must be migrated to OpenTofu.**

This likely means the automatic migration to OpenTofu was canceled due to destructive changes detected in the cluster state.

Verify the following:

- Current Terraform version:

```
d8 k -n d8-system exec -it deployments/terraform-state-exporter -C exporter -- terraform version
```

- Version in the Terraform state:

```
d8 k -n d8-system get secret d8-cluster-terraform-state -o json | jq -r '.data["cluster-tf-state.json"] | @base64d | fromjson | .terraform_version'
```

- Check for destructive changes:

```
d8 k exec -it deploy/terraform-state-exporter -n d8-system -- dhctl terraform check
```

To resolve the issue and migrate to OpenTofu manually, follow these steps:

1. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run:

```
dhctl converge
```

This should resolve all destructive changes in the cluster.

2. After that, the `terraform-auto-converger` Pod should complete the migration automatically.

If the automatic migration doesn't happen, use the `install` container with the current Deckhouse release and run the following command to enforce migration of the Terraform state to OpenTofu:

```
dhctl converge-migration
```

If this alert appears in a cluster created by Deckhouse Commander, it means the cluster has been updated to a new Deckhouse version without approving state changes (for example, destructive changes).

To resolve this, follow these steps:

1. In Deckhouse Commander, click **Delete manually** to temporarily remove the cluster from control.
2. In the cluster, remove the `commander-agent` module using the following command:

```
d8 k delete mc commander-agent
```

3. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run the following command to resolve all destructive changes in the cluster:

```
dhctl converge
```

4. In Deckhouse Commander, click **Attach** to reattach the cluster.

- D8NodeCgroupV2NotSupported CE S8

▼ **Node `NODE_NAME` is using deprecated cgroup v1 - migration to v2 required.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` does not support cgroup v2. Starting from Kubernetes 1.35, cgroup v1 is deprecated. To migrate to cgroup v2, please upgrade the kernel to version 5.8 or newer. For more information, please refer to the documentation.

- D8NodeContainerdV2NotSupported CE S8

▼ **Node `NODE_NAME` is using deprecated containerd v1 - migration to v2 required.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` does not support containerd v2. Starting from Kubernetes 1.37, containerd v1.y will no longer be supported. Please check the requirements for containerd v2 and schedule your migration plan.

The exact reason is listed in the `node.deckhouse.io/containerd-v2-err` annotation on the node.

If the reason is `erofs` and the node runs Ubuntu in AWS, the module is shipped in a separate `linux-modules-extra` package. DKP installs it automatically, but for some kernels, mostly outdated ones, the package is no longer published in the APT repository. In that case, the installation is skipped and a warning is written to the node logs. The problem has been confirmed for the `5.15.0-1028-aws`, `6.8.0-1024-aws`, and `6.8.0-1029-aws` kernels. Switch the node group to a newer AMI or kernel version that includes `erofs`.

- D8NodeGroupsNotUpdating CE S8

▼ **NodeGroup `NODE_GROUP_NAME` is not handling the update correctly.**

There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. Although Nodes have detected the update, none of them have received approval to start the update process.

Most likely, there is a problem with the `update_approval` hook of the `node-manager` module.

- D8NodesNotUpdating CE S7

▼ **Node `NODE_NAME` cannot complete the update.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received approval, started the update, and encountered a step that could cause downtime. The

update manager (the `update_approval` hook of the `node-group` module) granted downtime approval, but no success message was received, which indicates that the update has not completed.

To investigate the details, view Bashible logs on the Node:

```
journalctl -fu bashible
```

- D8NodesNotUpdating CE S8

- ▼ **Node `NODE_NAME` cannot complete the update.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, requested and received approval, but failed to complete the update.

To investigate the details, view Bashible logs on the Node:

```
journalctl -fu bashible
```

- D8NodesNotUpdating CE S9

- ▼ **Node `NODE_NAME` is not updating.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has a pending update but it's neither receiving it nor attempting to.

Most likely, Bashible is not handling the update correctly. It should annotate the Node with `update.node.deckhouse.io/waiting-for-approval` before the update can proceed.

Options to investigate the details:

- Check the expected configuration checksum for the NodeGroup:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o jsonpath={.data.NODE_GROUP_NAME} | base64 -d
```

- Check the current configuration checksum on the Node:

```
d8 k get node NODE_NAME -o jsonpath='{.metadata.annotations.node\.deckhouse\.io/configuration-checksum}'
```

- View Bashible logs on the Node:

```
journalctl -fu bashible
```

- D8NodesUnmanaged CE S9

- ▼ **The `NODE_NAME` Node is not managed by the node-manager module.**

The `NODE_NAME` Node is not managed by the `node-manager` module.

To resolve this issue, follow the instructions how to clean up the node and add it to the cluster.

- D8NodeUpdateStuckWaitingForDisruptionApproval CE S8

- ▼ **Node `NODE_NAME` cannot obtain disruption approval.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received initial approval, and started the update. However, it reached a stage that could cause downtime and was unable to obtain disruption approval. A disruption approval is normally issued automatically by the `update_approval` hook of the `node-manager` module.

To resolve this issue, investigate why the approval could not be granted to proceed with the update.

- D8ProblematicNodeGroupConfiguration CE S8

- ▼ **Node `NODE_NAME` cannot begin the update.**

There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. However, Node `NODE_NAME` cannot begin the update.

The Node is missing the `node.deckhouse.io/configuration-checksum` annotation, which may indicate that its bootstrap process did not complete correctly.

Troubleshooting options:

- Check the `cloud-init log (/var/log/cloud-init-output.log)` on the node.
- Check the `NodeGroupConfiguration` resource associated with the `NODE_GROUP_NAME` `NodeGroup` for potential issues.
- **EarlyOOMPodsNotReady CE S8**
▼ Pod `POD_NAME` has detected an unavailable PSI subsystem.

The Pod `POD_NAME` detected that the Pressure Stall Information (PSI) subsystem is unavailable.

For details, check the logs:

```
d8 k -n d8-cloud-instance-manager logs POD_NAME
```

Troubleshooting options:

- Upgrade the Linux kernel to version 4.20 or higher.
- Enable the Pressure Stall Information.
- Disable early OOM.
- **NodeGroupHasStaticInternalNetworkCIDRsField CE S9**
▼ NodeGroup `NAME` uses deprecated field `spec.static.internalNetworkCIDRs`.

The `spec.static.internalNetworkCIDRs` field is deprecated and has been moved to the static cluster configuration.

To resolve this issue, delete the field from `NodeGroup NAME`. This is safe, as the setting has already been migrated automatically.

- **NodeGroupMasterTaintIsAbsent CE S4**
▼ The master NodeGroup is missing the required control-plane taint.

The `master` `NodeGroup` doesn't have the `node-role.kubernetes.io/control-plane:NoSchedule` taint. This may indicate a misconfiguration where control-plane nodes can run non-control-plane Pods.

To resolve the issue, add the following to the `master` `NodeGroup` spec:

```
nodeTemplate:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/control-plane
```

Note that the taint `key: node-role.kubernetes.io/master` is deprecated and has no effect starting from Kubernetes 1.24.

- **NodeGroupReplicasUnavailable CE S7**
▼ NodeGroup `NODE_GROUP_NAME` has no available instances.

This probably means that `machine-controller-manager` is unable to create `Machines` using the cloud provider module.

Possible causes:

1. Cloud provider resource limits.
2. No access to the cloud provider API.
3. Misconfiguration of the cloud provider or instance class.
4. Problems with bootstrapping the Machine.

Recommended actions:

1. Check the status of the `NodeGroup`:

```
d8 k get ng NODE_GROUP_NAME -O yaml
```

Look for errors in the `.status.lastMachineFailures` field.

2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

```
d8 k get instances
```

3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap

status:

```
d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in description) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- **NodeGroupReplicasUnavailable CE S8**

- ▼ **Too many unavailable instances in the `NODE_GROUP_NAME` NodeGroup.**

The number of simultaneously unavailable instances in the `NODE_GROUP_NAME` NodeGroup exceeds the allowed threshold. This may indicate that the autoscaler has provisioned too many nodes at once. Check the state of the Machine in the cluster. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

Possible causes:

1. Cloud provider resource limits.
2. No access to the cloud provider API.
3. Misconfiguration of the cloud provider or instance class.
4. Problems with bootstrapping the Machine.

Recommended actions:

1. Check the status of the NodeGroup:

```
d8 k get ng NODE_GROUP_NAME -o yaml
```

Look for errors in the `.status.lastMachineFailures` field.

2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

```
d8 k get instances
```

3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

```
d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in description) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- **NodeGroupReplicasUnavailable CE S8**

- ▼ **NodeGroup `NODE_GROUP_NAME` has unavailable instances.**

There are VALUE unavailable instances in the NodeGroup `NODE_GROUP_NAME`. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

Possible causes:

1. Cloud provider resource limits.
2. No access to the cloud provider API.
3. Misconfiguration of the cloud provider or instance class.
4. Problems with bootstrapping the Machine.

Recommended actions:

1. Check the status of the NodeGroup:

```
d8 k get ng NODE_GROUP_NAME -O yaml
```

Look for errors in the `.status.lastMachineFailures` field.

2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

```
d8 k get instances
```

3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

```
d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in description) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeRequiresDisruptionApprovalForUpdate CE S8

- ▼ **Node `NODE_NAME` requires disruption approval to proceed with the update.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, received initial approval, and started the update. However, it encountered a stage that may cause downtime and requires manual disruption approval, because the NodeGroup is configured with `disruptions.approvalMode: Manual`.

To resolve the issue, ensure the Node is ready for unsafe updates (drained) and grant disruption approval by annotating the Node with `update.node.deckhouse.io/disruption-approved=`.

Caution:

- Nodes in manual mode aren't drained automatically.
- Do not drain master nodes.

1. To drain the Node and grant the update approval, run the following command:

```
d8 k drain NODE_NAME --delete-emptydir-data --ignore-daemonsets --force=true &&
d8 k annotate node NODE_NAME update.node.deckhouse.io/disruption-approved=
```

2. Uncordon the Node once the update is complete and the annotation `update.node.deckhouse.io/approved` is removed:

```
while d8 k get node NODE_NAME -o json | jq -e '.metadata.annotations |
has("update.node.deckhouse.io/approved")' > /dev/null; do sleep 1; done
d8 k uncordon NODE_NAME
```

If the NodeGroup has multiple Nodes, repeat this process for each one, since only one Node is updated at a time. Consider temporarily switching to automatic disruption approval (`disruptions.approvalMode: Automatic`).

- NodeStuckInDraining CE S6

- ▼ **Node `NODE_NAME` is stuck in draining.**

Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` is stuck in the draining process.

To get more details, run the following:

```
d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=DrainFailed --sort-by='.metadata.creationTimestamp'
```

Error message: MESSAGE_CONTENTS

- NodeStuckInDrainingForDisruptionDuringUpdate CE S6
▼ **Node NODE_NAME is stuck in draining.**

Node NODE_NAME in NodeGroup NODE_GROUP_NAME has detected a new update, requested and received approval, started the update, and reached a step that could cause downtime. It is currently stuck in the draining process while waiting for automatic disruption approval.

To get more details, run the following:

```
d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=ScaleDown --sort-by='.metadata.creationTimestamp'
```

- StandbyHolderDeploymentReplicasUnavailable CE S5
▼ **No available replicas in Deployment standby-holder-NODE_GROUP_NAME.**

Deckhouse has detected that there are no available replicas in the standby-holder-NODE_GROUP_NAME Deployment in the d8-cloud-instance-manager namespace.

Check the Deployment and associated Pods for issues.

- UnmetCloudConditions CE S6
▼ **Deckhouse update is unavailable due to unmet cloud provider conditions.**

Deckhouse has detected that some cloud provider-specific conditions have not been met. Until these issues are resolved, updating to the new Deckhouse release is not possible.

Troubleshooting details:

- Name: NAME
- Message: MESSAGE_CONTENTS

Модуль okmeter

- D8OkmeterAgentPodIsNotReady CE S6

Okmeter agent is not Ready.

Модуль openvpn

- OpenVPNClientCertificateExpired CE S4
▼ **OpenVPN client certificate expired for XXX**

The OpenVPN client certificate for XXX has expired.

Check the certificate information and make sure it actually expires:

```
d8 k -n d8-openvpn get secrets -l name="XXX" -o jsonpath='{.items[0].data.tls.crt}' | base64 -d | openssl x509 -text -noout
```

Renew or delete the expired certificate. Check the documentation for details.

- OpenVPNClientCertificateExpired CE S5
▼ **OpenVPN client certificate expires for XXX in less than 7 days.**

The OpenVPN client certificate for XXX is set to expire in VALUE days.

Renewal of certificate required. Check the documentation for details.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets -l name=XXX -o jsonpath='{.items[0].data.tls.crt}' | base64 -d | openssl x509 -text -noout
```

- OpenVPNClientCertificateExpired CE S6
▼ **OpenVPN client certificate expires for XXX in less than 30 days.**

The OpenVPN client certificate for XXX is set to expire in VALUE days.

Renewal of certificate required. Check the documentation for details.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets -l name=xxx -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

- OpenVPNServerCACertificateExpired CE S4
▼ **OpenVPN CA certificate has expired.**

OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets openvpn-pki-ca -o jsonpath='{.data.tls\.crt}' | base64 -d
| openssl x509 -text -noout
```

The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiresTomorrow CE S4
▼ **OpenVPN CA certificate will expire tomorrow**

OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets openvpn-pki-ca -o jsonpath='{.data.tls\.crt}' | base64 -d
| openssl x509 -text -noout
```

The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiringInAWeek CE S5
▼ **OpenVPN CA certificate expires in less than 7 days.**

The OpenVPN CA certificate is set to expire in VALUE days.

Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets -l name=server -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

- OpenVPNServerCACertificateExpiringSoon CE S5
▼ **OpenVPN CA certificate expires in less than 30 days.**

The OpenVPN CA certificate is set to expire in VALUE days.

Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets -l name=server -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

- OpenVPNServerCertificateExpired CE S4
▼ **OpenVPN server certificate has expired.**

The OpenVPN server certificate has expired.

OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

Check the certificate information and make sure it actually expires.

```
d8 k -n d8-openvpn get secrets -l name=server -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

- OpenVPNServerCertificateExpiresTomorrow CE S4
▼ **OpenVPN server certificate will expire tomorrow.**

The OpenVPN server certificate is set to expire in less than 1 day.

OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

Check the certificate information and make sure it actually expires:

```
d8 k -n d8-openvpn get secrets -l name=server -o jsonpath='{.items[0].data.tls\.crt}' |
base64 -d | openssl x509 -text -noout
```

Модуль operator-prometheus

- D8PrometheusOperatorPodIsNotReady CE S7
▼ **The prometheus-operator Pod is NOT Ready.**

As a result, new Prometheus, PrometheusRules, and ServiceMonitor configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

Troubleshooting steps:

1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

2. Examine the Pod's to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

- D8PrometheusOperatorPodIsNotRunning CE S7
▼ **The prometheus-operator Pod is NOT Running.**

As a result, new Prometheus, PrometheusRules, and ServiceMonitor configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

Troubleshooting steps:

1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

2. Examine the Pod's to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

- D8PrometheusOperatorTargetAbsent CE S7
▼ **Prometheus-operator target is missing in Prometheus.**

As a result, new Prometheus, PrometheusRules, and ServiceMonitor configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

To resolve the issue, analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

- D8PrometheusOperatorTargetDown CE S8
▼ **Prometheus is unable to scrape prometheus-operator metrics.**

The prometheus-operator Pod is unavailable.

As a result, new Prometheus, PrometheusRules, and ServiceMonitor configurations cannot be

applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

Troubleshooting steps:

1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

2. Examine the Pod's to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

Модуль prometheus

- D8GrafanaDeploymentReplicasUnavailable CE S6
▼ **One or more Grafana Pods are NOT Running.**

The number of Grafana replicas is lower than the specified number.

The Deployment is in the `MinimumReplicasUnavailable` state.

Troubleshooting options:

- To check the Deployment's status:

```
d8 k -n d8-monitoring get deployment grafana-v10 -o json | jq .status
```

- To check a Pod's status:

```
d8 k -n d8-monitoring get pods -l app=grafana-v10 -o json | jq '.items[] | {(.metadata.name):.status}'
```

- D8GrafanaDeprecatedCustomDashboardDefinition CE S9
▼ **Deprecated ConfigMap for Grafana dashboards detected.**

The ConfigMap `grafana-dashboard-definitions-custom` has been found in the `d8-monitoring` namespace. This indicates that a deprecated method for registering custom dashboards in Grafana is used.

This method is no longer supported.

Migrate to using the custom GrafanaDashboardDefinition resource instead.

- D8GrafanaPodIsNotReady CE S6

The Grafana Pod is NOT Ready.

- D8GrafanaPodIsRestartingTooOften CE S9
▼ **Excessive Grafana restarts detected.**

Grafana has restarted VALUE times in the last hour.

Frequent restarts indicate a problem. Grafana is expected to run continuously without interruption.

To investigate the issue, check the logs:

```
d8 k -n d8-monitoring logs -f -l app=grafana-v10 -C grafana
```

- D8GrafanaTargetAbsent CE S6
▼ **Grafana target is missing in Prometheus.**

Grafana visualizes metrics collected by Prometheus. Grafana is critical for some tasks, such as monitoring the state of applications and the cluster as a whole. Additionally, Grafana's unavailability can negatively impact users who actively use it in their work.

The recommended course of action:

1. Check the availability and status of Grafana Pods:

```
d8 k -n d8-monitoring get pods -l app=grafana-v10
```

2. Check the availability of the Grafana Deployment:

```
d8 k -n d8-monitoring get deployment grafana-v10
```

3. Examine the status of the Grafana Deployment:

```
d8 k -n d8-monitoring describe deployment grafana-v10
```

- D8GrafanaTargetDown CE S6

Prometheus is unable to scrape Grafana metrics.

- D8PrometheusConfigReloadFailed CE S5

- ▼ **Prometheus configuration reload failed.**

The Prometheus configuration reload has failed in the `d8-monitoring` namespace.

This usually happens when:

- Invalid scrape configuration (e.g., from ServiceMonitor/PodMonitor resources).
- Invalid RemoteWrite configuration.

Prometheus will continue running with the old configuration, but new rules, targets, or remote write endpoints will not be active. Troubleshooting steps:

1. Check the Prometheus container logs for configuration errors:

```
d8 k -n d8-monitoring logs prometheus-main-0 -C prometheus
```

2. If using remote write, verify the remote write configuration.

Note: In HA configurations with multiple Prometheus replicas, check `prometheus-main-1` logs as well.

- D8PrometheusLongtermFederationTargetDown CE S5

- ▼ **Prometheus-longterm cannot scrape Prometheus.**

The `prometheus-longterm` instance is unable to scrape the `/federate` endpoint from Prometheus.

Troubleshooting options:

- Check the `prometheus-longterm` logs.
- Open the corresponding web UI to check scrape errors.

- D8PrometheusLongtermTargetAbsent CE S7

- ▼ **Prometheus-longterm target is missing in Prometheus.**

The `prometheus-longterm` component is used to display historical monitoring data and is not crucial. However, its extended downtime may prevent access to statistics.

This issue is often caused by problems with disk availability. For example, if the disk cannot be mounted to a Node.

Troubleshooting steps:

1. Check the StatefulSet status:

```
d8 k -n d8-monitoring describe statefulset prometheus-longterm
```

2. Inspect the PersistentVolumeClaim (if used):

```
d8 k -n d8-monitoring describe pvc prometheus-longterm-db-prometheus-longterm-0
```

3. Inspect the Pod's state:

```
d8 k -n d8-monitoring describe pod prometheus-longterm-0
```

- D8TricksterTargetAbsent CE S5

▼ Trickster target is missing in Prometheus.

The following modules use the Trickster component:

- `prometheus-metrics-adapter`: Its unavailability means horizontal pod autoscaling (HPA) is not working, and you cannot view resource consumption using `d8 k`.
- `vertical-pod-autoscaler`: Short-term unavailability for this module is tolerable, as VPA looks at the consumption history for 8 days.
- `grafana`: All dashboards use Trickster by default to cache Prometheus queries. You can retrieve data directly from Prometheus (bypassing the Trickster). However, this may lead to high memory usage by Prometheus and cause unavailability.

Troubleshooting steps:

1. Inspect the Deployment's stats:

```
d8 k -n d8-monitoring describe deployment trickster
```

2. Inspect the Pod's stats:

```
d8 k -n d8-monitoring describe pod -l app=trickster
```

3. Trickster often becomes unavailable due to Prometheus issues, since its `readinessProbe` depends on Prometheus being accessible.

Make sure Prometheus is running:

```
d8 k -n d8-monitoring describe pod -l app.kubernetes.io/name=prometheus,prometheus=main
```

- D8TricksterTargetAbsent CE S5

▼ Trickster target is missing in Prometheus.

The following modules use the Trickster component:

- `prometheus-metrics-adapter`: Its unavailability means horizontal pod autoscaling (HPA) is not working, and you cannot view resource consumption using `d8 k`.
- `vertical-pod-autoscaler`: Short-term unavailability for this module is tolerable, as VPA looks at the consumption history for 8 days.
- `grafana`: All dashboards use Trickster by default to cache Prometheus queries. You can retrieve data directly from Prometheus (bypassing the Trickster). However, this may lead to high memory usage by Prometheus and cause unavailability.

Troubleshooting steps:

1. Inspect the Deployment's stats:

```
d8 k -n d8-monitoring describe deployment trickster
```

2. Inspect the Pod's stats:

```
d8 k -n d8-monitoring describe pod -l app=trickster
```

3. Trickster often becomes unavailable due to Prometheus issues, since its `readinessProbe` depends on Prometheus being accessible.

Make sure Prometheus is running:

```
d8 k -n d8-monitoring describe pod -l app.kubernetes.io/name=prometheus,prometheus=main
```

- DeckhouseModuleUseEmptyDir CE S9

▼ Deckhouse module MODULE_NAME is using emptyDir for storage.

The Deckhouse module `MODULE_NAME` is using `emptyDir` as its storage.

If the associated Pod is removed from the node for any reason, the data in the `emptyDir` is deleted permanently. Consider using persistent storage if data durability is critical for the module.

- GrafanaDashboardAlertRulesDeprecated CE S8

▼ Deprecated Grafana alerts detected.

Before upgrading to Grafana 10, migrate outdated alerts from Grafana to an external

alertmanager (or `exporter-alertmanager` stack).

To list all deprecated alert rules, use the following expression:

```
sum by (dashboard, panel, alert_rule)
(d8_grafana_dashboards_deprecated_alert_rule) > 0
```

Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated resources are migrated.

- GrafanaDashboardPanelIntervalDeprecated CE S8
▼ **Deprecated Grafana panel intervals detected.**

Before upgrading to Grafana 10, update outdated panel expressions that use `$interval_rv`, `interval_sx3`, or `interval_sx4` to the new variable `$_rate_interval`.

To list all deprecated panel intervals, use the following expression:

```
sum by (dashboard, panel, interval) (d8_grafana_dashboards_deprecated_interval) > 0
```

Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated intervals are removed.

- GrafanaDashboardPluginsDeprecated CE S8
▼ **Deprecated Grafana plugins detected.**

Before upgrading to Grafana 10, make sure that the currently installed plugins will work correctly with Grafana 10.

To list all potentially outdated plugins, use the following expression:

```
sum by (dashboard, panel, plugin) (d8_grafana_dashboards_deprecated_plugin) > 0
```

The `flant-statusmap-panel` plugin is deprecated and no longer supported. It's recommended that you migrate to the state timeline plugin instead.

Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated resources are migrated.

- K8STooManyNodes CE S7
▼ **Node count is approaching the maximum allowed.**

The cluster is currently running VALUE nodes, which is close to the maximum of XXX allowed nodes.

- PrometheusDirectAccessDeprecated CE S6
▼ **Deprecated Ingress with direct Prometheus access found in cluster.**

The DKP cluster has an Ingress that allows direct access to Prometheus with client certificate authentication, for example, for the external metrics access. This approach is deprecated and the direct access to Prometheus will not be possible in the future DKP releases.

To resolve this issue, remove the deprecated Ingress and use the `observability` module to manage Prometheus metrics access.

Refer to the module documentation for instructions.

- PrometheusDiskUsage CE S4
▼ **Prometheus disk usage exceeds 95%.**

Deckhouse has detected that the Prometheus disk is over 95% full.

To check the current usage, run the following command:

```
d8 k -n NAMESPACE exec -ti POD_NAME -C prometheus -- df -PBG /prometheus
```

Consider expanding disk size following the guide.

- PrometheusLongtermRotatingEarlierThanConfiguredRetentionDays CE S4
▼ **Prometheus-longterm data is rotating earlier than the configured retention period.**

The `prometheus-longterm` instance is rotating data earlier than specified by the

The `prometheus-longterm` instance is rotating data earlier than specified by the `longtermRetentionDays` parameter.

Troubleshooting options:

- Increase the Prometheus disk size.
- Reduce the number of metrics collected.
- Lower the `longtermRetentionDays` parameter in the module configuration.
- PrometheusMainRotatingEarlierThanConfiguredRetentionDays CE S4
 - ▼ **Prometheus-main data is rotating earlier than the configured retention period.**

The `prometheus-main` instance is rotating data earlier than specified by the `retentionDays` parameter.

Troubleshooting options:

- Increase the Prometheus disk size.
- Reduce the number of metrics collected.
- Lower the `retentionDays` parameter in the module configuration.
- PrometheusRemoteWriteEndpointUnavailable CE S7
 - ▼ **Prometheus remote write endpoint is unavailable.**

Prometheus cannot send metrics to the remote write endpoint `XXX`.

To troubleshoot you can:

1. Get list of all Prometheus pods:

```
d8 k -n d8-monitoring get pods -l "app.kubernetes.io/name=prometheus"
```

Review logs of each prometheus pod to find the detailed error information:

```
d8 k -n NAMESPACE logs `pod-name` -f
```

2. Get logs from all Prometheus pods to find detailed information:

```
d8 k -n NAMESPACE logs -l "app.kubernetes.io/name=prometheus"
```

- PrometheusScrapeConfigDeclarationDeprecated CE S8
 - ▼ **AdditionalScrapeConfigs from Secrets will be deprecated soon.**

Defining additional scrape configurations using Secrets will be deprecated in `prometheus-operator` versions later than v0.65.1.

Migrate to using the `ScrapeConfig` custom resource instead.

- PrometheusServiceMonitorDeprecated CE S8
 - ▼ **Deprecated Prometheus ServiceMonitor detected.**

The Kubernetes cluster uses the modern `EndpointSlice` network mechanism, but the `ServiceMonitor` `NAMESPACE/NAME` still uses relabeling rules based on the old `Endpoints` mechanism (starting with `__meta_kubernetes_endpoints_`).

Support for the old relabeling rules will be removed in a future Deckhouse release (1.60). To avoid issues, update your `ServiceMonitor` to use `EndpointSlice`-based relabeling.

Update relabeling rules as follows:

```
__meta_kubernetes_endpoints_name -> __meta_kubernetes_endpointslice_name
__meta_kubernetes_endpoints_label_XXX -> __meta_kubernetes_endpointslice_label_XXX
__meta_kubernetes_endpoints_labelpresent_XXX -> __meta_kubernetes_endpointslice_labelpresent_XXX
__meta_kubernetes_endpoints_annotation_XXX -> __meta_kubernetes_endpointslice_annotation_XXX
__meta_kubernetes_endpoints_annotationpresent_XXX ->
__meta_kubernetes_endpointslice_annotationpresent_XXX
__meta_kubernetes_endpoint_node_name ->
__meta_kubernetes_endpointslice_endpoint_topology_kubernetes_io_hostname
__meta_kubernetes_endpoint_ready -> __meta_kubernetes_endpointslice_endpoint_conditions_ready
__meta_kubernetes_endpoint_port_name -> __meta_kubernetes_endpointslice_port_name
__meta_kubernetes_endpoint_port_protocol -> __meta_kubernetes_endpointslice_port_protocol
__meta_kubernetes_endpoint_address_target_kind ->
__meta_kubernetes_endpointslice_address_target_kind
__meta_kubernetes_endpoint_address_target_name ->
__meta_kubernetes_endpointslice_address_target_name
```

- TargetDown CE S5
 - ▼ **A target is down.**

Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetDown CE S6
▼ **A target is down.**

Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetDown CE S7
▼ **A target is down.**

Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetSampleLimitExceeded CE S6
▼ **Scrapes are exceeding the sample limit.**

One or more targets are down because the Prometheus sample limit was exceeded during a scrape.

- TargetSampleLimitExceeded CE S7
▼ **Sampling limit is nearly reached.**

The target is close to exceeding the Prometheus's sampling limit. Less than 10% of the allowed samples are left.

Модуль secret-copier

- D8SecretCopierDeprecatedLabels CE S9
▼ **Obsolete antiopa_secret_copier=yes label detected.**

The `secret-copier` module has changed its labeling approach for original secrets in the `default` namespace.

The label `antiopa-secret-copier: "yes"` is deprecated and will be removed soon.

To resolve this issue, replace the label `antiopa-secret-copier: "yes"` with `secret-copier.deckhouse.io/enabled: ""` for all secrets used by the `secret-copier` module in the `default` namespace.

Модуль terraform-manager

- D8TerraformStateExporterClusterStateChanged CE S8
▼ **Terraform-state-exporter cluster state change detected.**

The current Kubernetes cluster state is `STATUS_REFERENCE` compared to the Terraform state.

It is important to reconcile the states.

Troubleshooting steps:

1. View the differences:

```
dhctl terraform check
```

2. Apply the necessary changes to bring the cluster in sync:

```
dhctl converge
```

- D8TerraformStateExporterClusterStateError CE S8
▼ **Terraform-state-exporter cluster state error.**

The `terraform-state-exporter` can't check difference between the Kubernetes cluster state and the Terraform state.

That was likely caused by `terraform-state-exporter` failing to run Terraform with the current state and configuration.

Troubleshooting steps:

1. View the differences:

```
dhctl terraform check
```

2. Apply the necessary changes to bring the cluster in sync:


```
dhctl converge
```

- D8TerraformStateExporterHasErrors CE S8
▼ **Terraform-state-exporter has encountered errors.**

Errors occurred during the operation of the `terraform-state-exporter`.

To get more details, check the Pod logs:

```
d8 k -n d8-system logs -l app=terraform-state-exporter -C exporter
```

- D8TerraformStateExporterNodeStateChanged CE S8
▼ **Terraform-state-exporter node state change detected.**

The current state of node `NODE_GROUP_NAME/NAME` is `STATUS_REFERENCE` compared to the Terraform state.

It is important to reconcile the states.

Troubleshooting steps:

1. View the differences:

```
dhctl terraform check
```

2. Apply the necessary changes to bring the cluster in sync:

```
dhctl converge
```

- D8TerraformStateExporterNodeStateError CE S8
▼ **Terraform-state-exporter node state error.**

The `terraform-state-exporter` can't check the difference between the node `NODE_GROUP_NAME/NAME` state and the Terraform state.

Probably, it occurred because `terraform-manager` had failed to run Terraform with the current state and configuration.

Troubleshooting steps:

1. View the differences:

```
dhctl terraform check
```

2. Apply the necessary changes to bring the cluster in sync:

```
dhctl converge
```

- D8TerraformStateExporterNodeTemplateChanged CE S8
▼ **Terraform-state-exporter node template change detected.**

The `terraform-state-exporter` has detected a mismatch between the node template in the cluster provider configuration and the one specified in the NodeGroup `NAME`.

Node template is `STATUS_REFERENCE`.

Troubleshooting steps:

1. View the differences:

```
dhctl terraform check
```

2. Adjust NodeGroup settings to fix the issue or bring the cluster in sync via the following command:

```
dhctl converge
```

- D8TerraformStateExporterPodIsNotReady CE S8
▼ **Terraform-state-exporter Pod is not Ready.**

The `terraform-state-exporter` cannot check the difference between the actual Kubernetes

cluster state and the Terraform state.

To resolve the issue, check the following:

1. Deployment description:

```
d8 k -n d8-system describe deployment terraform-state-exporter
```

2. Pod status:

```
d8 k -n d8-system describe pod -l app=terraform-state-exporter
```

- D8TerraformStateExporterPodIsNotRunning CE S8
▼ **Terraform-state-exporter Pod is not Running.**

The `terraform-state-exporter` cannot check the difference between the actual Kubernetes cluster state and the Terraform state.

To resolve the issue, check the following:

1. Deployment description:

```
d8 k -n d8-system describe deployment terraform-state-exporter
```

2. Pod status:

```
d8 k -n d8-system describe pod -l app=terraform-state-exporter
```

- D8TerraformStateExporterTargetAbsent CE S8
▼ **Terraform-state-exporter target is missing in Prometheus.**

Prometheus cannot find the `terraform-state-exporter` target.

To investigate the details:

- Check the Pod status:

```
d8 k -n d8-system get pod -l app=terraform-state-exporter
```

- Check the container logs:

```
d8 k -n d8-system logs -l app=terraform-state-exporter -C exporter
```

- D8TerraformStateExporterTargetDown CE S8
▼ **Prometheus can't scrape terraform-state-exporter.**

Prometheus is unable to scrape metrics from the `terraform-state-exporter`.

To investigate the details:

- Check the Pod status:

```
d8 k -n d8-system get pod -l app=terraform-state-exporter
```

- Check the container logs:

```
d8 k -n d8-system logs -l app=terraform-state-exporter -C exporter
```

Модуль upmeter

- D8SmokeMiniNotBoundPersistentVolumeClaims CE S9
▼ **Smoke-mini has unbound or lost PersistentVolumeClaims.**

The PersistentVolumeClaim `PVC_NAME` is in the `STATUS` phase.

This indicates a problem with PersistentVolume provisioning.

To investigate the cause, check the PersistentVolumeClaim phase:

```
d8 k -n d8-upmeter get pvc PVC_NAME
```

If your cluster doesn't have a disk provisioning system, you can disable volume ordering for `smoke-mini` using the module settings.

- D8UpmeterAgentPodsNotReady CE S6

Upmeter agent is not Ready.

- D8UpmeterAgentReplicasUnavailable CE S6

▼ **One or more upmeter-agent Pods are NOT Running.**

Some `upmeter-agent` Pods are not in the Running state.

To investigate the details, do the following:

- Check the DaemonSet status:

```
d8 k -n d8-upmeter get daemonset upmeter-agent -o json | jq .status
```

- Check the Pod status:

```
d8 k -n d8-upmeter get pods -l app=upmeter-agent -o json | jq '.items[] | {(.metadata.name):.status}'
```

- D8UpmeterDiskUsage CE S5

▼ **Upmeter disk usage exceeds 80%.**

Disk usage for Upmeter has exceeded 80%.

The only way to resolve this is to recreate the PersistentVolumeClaim (PVC) in the following steps:

1. Save the PVC data if you need it.
2. Delete the PVC and restart the `upmeter` Pod:

```
d8 k -n d8-upmeter delete persistentvolumeclaim/data-upmeter-0 pod/upmeter-0
```

3. Check the status of the created PVC:

```
d8 k -n d8-upmeter get pvc
```

- D8UpmeterProbeGarbageConfigmap CE S9

▼ **Garbage ConfigMaps from the basic probe are not being cleaned up.**

Probe-generated ConfigMaps were found but not cleaned up as expected.

`upmeter-agent` should automatically delete ConfigMaps produced by the `control-plane/basic` probe. There should be no more ConfigMaps than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master nodeSelector), and they should be deleted within seconds.

This may indicate:

- A problem with the `kube-apiserver`.
- Stale ConfigMaps left by outdated `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check the `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? | select(.group=="control-plane" and .probe == "basic-functionality") | [.time, .level, .msg] | @tsv'
```

2. Ensure the control plane is operating normally.
3. Delete stale ConfigMaps manually:

```
d8 k -n d8-upmeter delete cm -l heritage=upmeter
```

- D8UpmeterProbeGarbageDeployment CE S9
▼ **Garbage Deployments from the controller-manager probe are not being cleaned up.**

The average number of probe-generated Deployments per `upmeter-agent` Pod is VALUE.

`upmeter-agent` should automatically delete Deployments created by the `control-plane/controller-manager` probe. There should not be more Deployments than master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

This may indicate:

- A problem with the `kube-apiserver`.
- Stale Deployments left by outdated `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check the `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
select(.group=="control-plane" and .probe == "controller-manager") |
[.time, .level, .msg] | @tsv'
```

2. Ensure the control plane is operating normally. Pay close attention to `kube-controller-manager`.
3. Delete stale Deployments manually:

```
d8 k -n d8-upmeter delete deployment -l heritage=upmeter
```

- D8UpmeterProbeGarbageNamespaces CE S9
▼ **Garbage namespaces from the namespace probe are not being cleaned up.**

The average number of probe-created namespaces per `upmeter-agent` Pod is VALUE.

`upmeter-agent` should automatically clean up namespaces created by the `control-plane/namespace` probe. There should not be more of these namespaces than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

This may indicate:

- A problem with the `kube-apiserver`.
- Stale namespaces left from older `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
select(.group=="control-plane" and .probe == "namespace") | [.time, .level,
.msg] | @tsv'
```

2. Ensure the control plane is operating normally.
3. Delete stale namespaces manually:

```
d8 k -n d8-upmeter delete ns -l heritage=upmeter
```

- D8UpmeterProbeGarbagePods CE S9
▼ **Garbage Pods from the scheduler probe are not being cleaned up.**

The average number of probe Pods per `upmeter-agent` Pod is VALUE.

`upmeter-agent` should automatically clean up Pods created by the `control-plane/scheduler` probe.

There should not be more of these Pods than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

This may indicate:

- A problem with the `kube-apiserver`.
- Stale Pods left from old `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
select(.group=="control-plane" and .probe == "scheduler") | [.time, .level,
.msg] | @tsv'
```

2. Ensure the control plane is operating normally.
3. Delete stale Pods manually:

```
d8 k -n d8-upmeter delete pods -l upmeter-probe=scheduler
```

- D8UpmeterProbeGarbagePodsFromDeployments CE S9

- ▼ **Garbage Pods from the controller-manager probe are not being cleaned up.**

A Pod from the `control-plane/controller-manager` probe was created more than 5 minutes ago.

`upmeter-agent` is expected to clean Deployments created by the `control-plane/controller-manager` probe, and `kube-controller-manager` should remove the associated Pods within seconds.

This may indicate:

- A problem with the `kube-apiserver` or `kube-controller-manager`.
- Stale Pods left from outdated `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check the `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
select(.group=="control-plane" and .probe == "controller-manager") |
[.time, .level, .msg] | @tsv'
```

2. Ensure the control plane is operating normally. Pay close attention to `kube-controller-manager`.
3. Delete stale Pods manually:

```
d8 k -n d8-upmeter delete pods -l upmeter-probe=controller-manager
```

- D8UpmeterProbeGarbageSecretsByCertManager CE S9

- ▼ **Garbage Secrets from the cert-manager probe are not being cleaned up.**

Probe-generated Secrets were found.

`upmeter-agent` should clean up certificates created by the `cert-manager` probe, and `cert-manager` in turn should clean up the associated Secrets. There should not be more of these Secrets than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master nodeSelector), and they should be deleted within seconds.

This may indicate:

- A problem with `kube-apiserver`, `cert-manager`, or `upmeter`.
- Stale Secrets left from older `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check `upmeter-agent` logs:

```
d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
select(.group=="control-plane" and .probe == "cert-manager") | [.time,
.level, .msg] | @tsv'
```

2. Check that the control plane and `cert-manager` are operating normally.

3. Delete stale certificates and Secrets manually:

```
d8 k -n d8-upmeter delete certificate -l upmeter-probe=cert-manager
d8 k -n d8-upmeter get secret -ojson | jq -r '.items[] | .metadata.name' | grep
upmeter-cm-probe | xargs -n 1 -- d8 k -n d8-upmeter delete secret
```

• D8UpmeterServerPodsNotReady CE S6

Upmeter server is not Ready.

• D8UpmeterServerPodsRestartingTooOften CE S9

▼ **Upmeter server is restarting too frequently.**

The `upmeter` server has restarted VALUE times in the last hour.

Frequent restarts may indicate an issue. It is expected to run continuously and collect availability episodes without interruption.

To investigate the cause, check the logs:

```
d8 k -n d8-upmeter logs -f upmeter-0 upmeter
```

• D8UpmeterServerReplicasUnavailable CE S6

▼ **One or more Upmeter server Pods are NOT Running.**

Some `upmeter` Pods are not in the Running state.

To investigate the details, do the following:

- Check the StatefulSet status:

```
d8 k -n d8-upmeter get statefulset upmeter -O json | jq .status
```

- Check the Pod status:

```
d8 k -n d8-upmeter get pods upmeter-0 -O json | jq '.items[] |
{(.metadata.name):.status}'
```

• D8UpmeterSmokeMiniMoreThanOnePVxPVC CE S9

▼ **Unnecessary smoke-mini PersistentVolumes detected in the cluster.**

The number of unnecessary `smoke-mini` PersistentVolumes (PVs) is VALUE.

These PVs should be deleted automatically when released.

Possible causes:

- The `smoke-mini` StorageClass may be set to `Retain` by default.
- There may be issues with the CSI driver or cloud storage integration.

These PVs do not contain valuable data and should be deleted.

To list all the PVs, run the following command:

```
d8 k get pv | grep disk-smoke-mini
```

• D8UpmeterTooManyHookProbeObjects CE S9

▼ **Too many UpmeterHookProbe objects in the cluster.**

The average number of `UpmeterHookProbe` objects per `upmeter-agent` Pod is VALUE, but it should always be exactly 1 per agent.

This likely happened because older `upmeter-agent` Pods left behind their `UpmeterHookProbe` objects during an Upmeter update or downscale.

Once the cause has been investigated, remove outdated objects and leave only the ones corresponding to currently running `upmeter-agent` Pods.

To view all `UpmeterHookProbe` objects, run the following command:

```
d8 k get upmeterhookprobes.deckhouse.io
```

Модуль user-authn

- D8DexAllTargetsDown CE S6

Prometheus is unable to scrape Dex metrics.

- D8DexAuthRequestsQuotaExhausted CE S3

▼ **Dex AuthRequest objects ResourceQuota is exhausted in namespace "d8-user-authn".**

The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` has been fully exhausted.

Dex is currently unable to create new AuthRequest custom resources, which will cause:

- Dex health checks to fail immediately.
- Authentication through external identity providers to stop working.
- Users unable to log in via OAuth/OIDC flows.

Immediate actions required:

1. Check current AuthRequest objects and clean up stale ones:

```
d8 k get authrequests.dex.coreos.com -n d8-user-authn --no-headers | wc -l
d8 k delete authrequests.dex.coreos.com --all -n d8-user-authn
```

2. Temporarily increase the ResourceQuota limit:

```
d8 k patch resourcequota authrequest-quota -n d8-user-authn --type='json' -p='[{"op":
"replace", "path": "/spec/hard/count/authrequests.dex.coreos.com",
"value": "20000"}]'
```

3. Investigate why Dex is not cleaning up AuthRequest objects automatically.

4. Review Dex configuration for AuthRequest TTL settings.

- D8DexAuthRequestsQuotaHigh CE S4

▼ **Dex AuthRequest objects quota usage is higher than 80% in namespace "d8-user-authn".**

The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` is approaching its limit.

Current usage: XXX%

When the quota is exhausted, Dex will be unable to create new AuthRequest custom resources, which may cause:

- Dex health checks to fail.
- Authentication through external identity providers (GitLab, GitHub, etc.) to stop working.

Investigation steps:

1. Check for stale AuthRequest objects that Dex should have cleaned up:

```
d8 k get authrequests.dex.coreos.com -n d8-user-authn
```

2. Review Dex configuration for AuthRequest TTL settings.
3. Consider temporarily increasing the ResourceQuota limit if needed.

Модуль user-authz

- D8UserAuthzReleaseSlow CE S6

▼ **The user-authz module release repeatedly takes more than 5 minutes.**

Applying the `user-authz` Helm release took more than 5 minutes on average over the last hour, and it was applied at least twice in that hour, so this is not the single slow pass of a cluster bootstrap or of the one-time migration to `user-authz-controller`. A release that does not finish within the module release timeout (20 minutes) fails, changes to `ClusterAuthorizationRule` and `AuthorizationRule` objects stop being applied and platform updates stall (the `D8DeckhouseCouldNotRunModule` alert fires in that case).

The release itself no longer grows with the number of rules: bindings are managed by `user-authz-controller`, and custom `ClusterRoles` are aggregated. A slow release therefore points

outside the module: an API server that answers slowly, an object stuck with a finalizer, a Deckhouse Pod short of CPU, or the release engine itself. Check what the release is waiting on:

```
d8 s queue list
d8 k -n d8-system logs deploy/deckhouse | grep '"module":"user-authz"'
```

If the cluster has just been updated to the release that moved the bindings to the controller, one slow pass is expected and does not raise this alert by itself; if it fires anyway, the pass is being retried. It is proportional to the number of bindings and can exceed the timeout on clusters with more than 5000 of them; the module FAQ describes what to do in that case.

- D8UserAuthzRenderSlow CE S7
 - ▼ **Rendering the user-authz module templates repeatedly takes more than 30 seconds.**

Rendering the templates of the `user-authz` module took more than 30 seconds on average over the last 30 minutes, across at least three renders; normally one render takes about a second. Rendering is CPU work inside the Deckhouse Pod and does not depend on the number of `ClusterAuthorizationRule` or `AuthorizationRule` objects, so the cause is the node, not the module: most often the Deckhouse Pod is short of CPU (CPU steal on a virtual machine, a node running at its limits, a bootstrap that starts every module at once).

A single slow render during a cluster bootstrap does not raise this alert by itself. If it fires, compare the module with the others and look at the node:

```
topk(10, increase(deckhouse_helm_operation_seconds_sum{operation="template"}[30m]) /
increase(deckhouse_helm_operation_seconds_count{operation="template"}[30m]))
rate(node_cpu_seconds_total{mode="steal"}[5m])
```

```
d8 k -n d8-system logs deploy/deckhouse | grep '"module":"user-authz"'
```

Параметры sysctl, настраиваемые платформой

Deckhouse автоматически настраивает и управляет рядом параметров работы ядра сервера, используя утилиту `sysctl`. Заданные параметры повышают сетевую пропускную способность, предотвращают нехватку ресурсов и оптимизируют управление памятью.

i При изменении этих параметров Deckhouse автоматически вернет их к значениям, перечисленным ниже.

Параметр	Значение, которое устанавливает Deckhouse	Описание
<code>/sys/block/*/queue/nr_requests</code>	256	Количество запросов в очереди для блочных устройств.
<code>/sys/block/*/queue/readahead_kb</code>	256	Объем дополнительных данных, которые ядро считывает с диска для ускорения чтения в дальнейшем.
<code>/sys/kernel/mm/transparent_hugepage/enabled</code>	never	Отключает Transparent HugePage.
<code>/sys/kernel/mm/transparent_hugepage/defrag</code>	never	Отключает дефрагментацию Transparent HugePage.
<code>/sys/kernel/mm/transparent_hugepage/use_zero_page</code>	0	Отключает использование нулевых huge-страниц.
<code>/sys/kernel/mm/transparent_hugepage/khugepaged/defrag</code>	0	Отключает дефрагментацию через khugepaged.
<code>/proc/sys/net/ipv4/conf/*/rp_filter</code>	0	Отключает «фильтрацию обратного пути» (reverse path filtering) для всех интерфейсов.

Параметр	Значение, которое устанавливает Deckhouse	Описание
<code>fs.file-max</code>	1000000	Максимальное количество открытых файлов.
<code>fs.inotify.max_user_instances</code>	5120	Максимальное количество экземпляров inotify.
<code>fs.inotify.max_user_watches</code>	524288	Максимальное количество файлов, отслеживаемых одним экземпляром inotify.
<code>fs.may_detach_mounts</code>	1	Разрешает отмонтировать файловую систему в режиме lazy unmounting.
<code>kernel.numa_balancing</code>	0	Запрещает автоматическую балансировку памяти с архитектурой NUMA.
<code>kernel.panic</code>	10 (0, если включен fencing)	Время в секундах, после которого произойдет перезагрузка узла при возникновении фатальной ошибки kernel panic. По умолчанию устанавливается значение 10. Если на узле включен режим fencing, устанавливается значение 0, тем самым предотвращая перезагрузку узла.
<code>kernel.panic_on_oops</code>	1	Разрешает системе активировать kernel panic при возникновении непредвиденной ошибки oops. Параметр необходим для корректной работы kubelet.
<code>kernel.pid_max</code>	2000000	Максимальное количество ID процессов (PID), которое можно назначить в системе.
<code>net.bridge.bridge-nf-call-arptables</code>	1	Разрешает фильтрацию трафика с помощью arptables. Параметр необходим для корректной работы kube-proxy.
<code>net.bridge.bridge-nf-call-ip6tables</code>	1	Разрешает фильтрацию трафика с помощью ip6tables. Параметр необходим для корректной работы kube-proxy.
<code>net.bridge.bridge-nf-call-iptables</code>	1	Разрешает фильтрацию трафика с помощью iptables. Параметр необходим для корректной работы kube-proxy.
<code>net.core.netdev_max_backlog</code>	5000	Максимальное количество пакетов в очереди на обработку.
<code>net.core.rmem_max</code>	16777216	Максимальный размер приемного буфера в байтах.
<code>net.core.somaxconn</code>	1000	Максимальное количество соединений в очереди.
<code>net.core.wmem_max</code>	16777216	Максимальный размер пересыльного буфера в байтах.

Параметр	Значение, которое устанавливает Deckhouse	Описание
<code>net.ipv4.conf.all.forwarding</code>	1	Разрешает перенаправление IPv4-пакетов между сетевыми интерфейсами. Равнозначно параметру <code>net.ipv4.ip_forward</code> .
<code>net.ipv4.ip_local_port_range</code>	"32768 61000"	Диапазон портов для исходящих TCP- и UDP-соединений.
<code>net.ipv4.neigh.default.gc_thresh1</code>	16384	Нижний порог количества ARP-записей, после которого система начнет удалять старые записи.
<code>net.ipv4.neigh.default.gc_thresh2</code>	28672	Средний порог количества ARP-записей, после которого система запустит очистку памяти.
<code>net.ipv4.neigh.default.gc_thresh3</code>	32768	Предельно допустимое количество ARP-записей.
<code>net.ipv4.tcp_max_syn_backlog</code>	8096	Максимальное количество SYN-соединений в очереди.
<code>net.ipv4.tcp_no_metrics_save</code>	1	Запрещает сохранение TCP-метрик закрытых соединений и их повторное использование для новых соединений.
<code>net.ipv4.tcp_rmem</code>	"4096 12582912 16777216"	Размеры приемного буфера для входящих TCP-пакетов в байтах: "<минимальный> <по умолчанию> <максимальный>".
<code>net.ipv4.tcp_slow_start_after_idle</code>	0	Запрещает использование окна перезагрузки (congestion window, CWND) и алгоритма медленного старта для TCP-соединений.
<code>net.ipv4.tcp_tw_reuse</code>	1	Разрешает повторное использование исходящих TCP-соединений в состоянии TIME-WAIT.
<code>net.ipv4.tcp_wmem</code>	"4096 12582912 16777216"	Размеры пересылочного буфера для исходящих TCP-пакетов в байтах: "<минимальный> <по умолчанию> <максимальный>".
<code>net.netfilter.nf_conntrack_max</code>	<кол-во ядер * 131072> или 524288	Максимальное количество отслеживаемых соединений в таблице conntrack. Рассчитывается по формуле: «количество выделенных CPU-ядер» * 131072, но не менее 524288.
<code>net.nf_conntrack_max</code>	<кол-во ядер * 131072> или 524288	Максимальное количество отслеживаемых соединений в таблице conntrack для старых версий ядра. Рассчитывается по формуле: «количество выделенных CPU-ядер» * 131072, но не менее 524288.

Параметр	Значение, которое устанавливает Deckhouse	Описание
<code>vm.dirty_background_ratio</code>	5	Доля системной памяти в процентах, которую допустимо занимать “грязными” страницами (dirty pages), прежде чем начнется запись на диск в асинхронном режиме.
<code>vm.dirty_expire_centisecs</code>	12000	Продолжительность периода в сотых долях секунды, пока “грязная” страница (dirty page) может оставаться в системной памяти, после чего она должна быть записана на диск.
<code>vm.dirty_ratio</code>	80	Доля системной памяти в процентах, которую допустимо занимать “грязными” страницами (dirty pages), прежде чем все процессы остановятся и будет выполнен сброс данных на диск.
<code>vm.min_free_kbytes</code>	131072	Минимальный объем свободной памяти в килобайтах, который резервируется ядром для выполнения критических операций.
<code>vm.overcommit_memory</code>	1	Разрешает избыточное выделение памяти (memory overcommitment).
<code>vm.swappiness</code>	0	Запрещает использование файла подкачки.

Дополнительные ресурсы

- Кастомные ресурсы модуля `node-manager`

Информация об используемом ПО

Список стороннего программного обеспечения, используемого в Deckhouse Kubernetes Platform (информация представлена на английском языке). Включает **только встроенные модули** (не включает модули, подключаемые из источников).

Список стороннего программного обеспечения используемого в конкретном модуле можно найти на странице описания в документации модуля.

cert-manager

Version: 1.20.0

A Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

Used in module cert-manager

Apache License 2.0

Chrony

chrony is a versatile implementation of the Network Time Protocol (NTP)

Used in module chrony

GNU General Public License version 2

Cilium

Version: 1.17.4

Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application workloads such as application containers or processes.

Used in module cni-cilium

Apache License 2.0

Cluster API

Version: 1.11.3

Kubernetes Cluster API for declarative cluster lifecycle management.

Used in module node-manager

Apache License 2.0

Cluster Autoscaler

Kubernetes autoscaler that automatically adjusts the size of the cluster.

Used in module node-manager

Apache License 2.0

CNI plugins

Version: 1.7.1

Some reference and example networking plugins, maintained by the CNI team.

Used in module cni-cilium

Apache License 2.0

CoreDNS

A DNS server/forwarder that chains plugins.

Used in module kube-dns

Apache License 2.0

Descheduler

Version: 0.35.1

A tool to auto-fix scheduling issues Kubernetes does not cover itself.

Used in module descheduler

Apache License 2.0

Dex

Version: 2.44.0

A Federated OpenID Connect Provider with pluggable connectors.

Used in module user-authn

Apache License 2.0

Etcd

Version: 3.6.10

A distributed reliable key-value store for the most critical data of a distributed system.

Used in module control-plane-manager

Apache License 2.0

Flannel

Version: 0.27.4

A simple and easy way to configure a layer 3 network fabric designed for Kubernetes.

Used in module cni-flannel

Apache License 2.0

Gardener Machine Controller Manager

Version: v0.36.0

An operator that manages VMs as a kubernetes custom resource.

Used in module node-manager

Apache License 2.0

Gatekeeper

Version: 3.22.0

Policy-based control for cloud native environments

Used in module admission-policy-engine

Apache License 2.0

Gops

Version: 0.3.27

A tool to list and diagnose Go processes currently running on your system.

Used in module cni-cilium

BSD 3-Clause License

Grafana

An open-source platform for monitoring and observability.

Used in module prometheus

GNU Affero General Public License v3.0

Hubble UI

Version: 0.13.2

Hubble UI is an open-source user interface for Cilium Hubble.

Used in module cilium-hubble

Apache License 2.0

Ingress NGINX Controller

Versions: 1.10.4, 1.12.1, 1.14.4

A ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

Used in module ingress-nginx

Apache License 2.0

Istio

Versions: 1.21.6, 1.25.2

An open platform to connect, manage, and secure microservices.

Used in module istio

Apache License 2.0

JQ

Version: 1.7.1

A lightweight and flexible command-line JSON processor.

Used in module deckhouse

Mixed (<https://github.com/stedolan/jq/blob/master/COPYING>)

keenalived

keepalived

Version: 2.2.7

Provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures.

Used in module keepalived

GNU General Public License v2.0

Kiali

Versions: 1.81.0, 2.7.1

Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

Used in module istio

Apache License 2.0

Kube-router

Version: 2.0.1

A turnkey solution for Kubernetes networking with aim to provide operational simplicity and high performance.

Used in module network-policy-engine

Apache License 2.0

kube-state-metrics

A simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.

Used in module monitoring-kubernetes

Apache License 2.0

Kubernetes

An open source system for managing containerized applications across multiple hosts.

Used in module control-plane-manager

Apache License 2.0

Kubernetes Autoscaler

Version: 1.6.0

A autoscaling-related components for Kubernetes.

Used in module vertical-pod-autoscaler

Apache License 2.0

Kubernetes Metrics Server

A scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

Used in module prometheus-metrics-adapter

Apache License 2.0

Libcap-ng

Version: 0.8.5

Libcap-ng is a library for Linux that makes using posix capabilities easy..

Used in module openvpn

GNU General Public License v2.0

Local-path-provisioner

Local Path Provisioner provides a way for the Kubernetes users to utilize the local storage in each node by creating hostPath based persistent volume on the node automatically.

Used in module local-path-provisioner

Apache License 2.0

Loki

Loki: like Prometheus, but for logs.

Used in module loki

GNU Affero General Public License v3.0

Memcached

High performance multithreaded event-based key/value cache store intended to be used in a distributed system.

Used in module prometheus

BSD-3-Clause license

MetalLB

Version: 0.15.2

A load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

Used in module metallb

Apache License 2.0

Mimir

A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

Used in module prometheus

AGPL-3.0 license

ModSecurity

Version: 3.0.14

ModSecurity is an open source, cross platform web application firewall (WAF) engine for Apache, IIS and Nginx.

Used in module ingress-nginx

Apache License 2.0

nlohmannJson

Version: 3.11.3

JSON for Modern C++

Used in module ingress-nginx

MIT License

Node exporter

Prometheus exporter for hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors.

Used in module monitoring-kubernetes

Apache License 2.0

OAuth2 Proxy

Version: 7.5.1

A reverse proxy and static file server that provides authentication using Providers (Google, GitHub, and others) to validate accounts by email, domain or group.

Used in module user-authn

MIT License

OpenKruise

Version: 898dc482a0cbf4fa0db214f952319aa5e1229152

OpenKruise is an extended component suite for Kubernetes, which mainly focuses on application automations, such as deployment, upgrade, ops and availability protection.

Used in module ingress-nginx

Apache License 2.0

OpenSSL

Version: 3.3.2

TLS/SSL and crypto library.

Used in module openvpn

Apache License 2.0

OpenVPN

Version: 2.6.12

A secure tunneling daemon.

Used in module openvpn

Mixed

OWASP CRS

Version: 4.10.0

The OWASP CRS is a set of generic attack detection rules for use with ModSecurity or compatible web application firewalls.

Used in module ingress-nginx

Apache License 2.0

Prometheus

A systems and service monitoring system.

Used in module prometheus

Apache License 2.0

Prometheus Operator

Kubernetes native deployment and management of Prometheus and related monitoring components.

Used in module operator-prometheus

Apache License 2.0

Prometheus Pushgateway

A tool to allow ephemeral and batch jobs to expose their metrics to Prometheus.

Used in module prometheus-pushgateway

Apache License 2.0

Promxy

Promxy is a prometheus proxy that makes many shards of prometheus appear as a single API endpoint to the user. This significantly simplifies operations and use of prometheus at scale (when you have more than one prometheus host).

Used in module prometheus

MIT License

Shell operator

Version: 1.9.3

A tool for running event-driven scripts in a Kubernetes cluster

Used in module deckhouse

Apache License 2.0

Terraform

A tool for building, changing, and versioning infrastructure safely and efficiently.

Used in module terraform-manager

Mozilla Public License 2.0

Trickster

A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

Used in module prometheus

Apache License 2.0

Vector

Vector is a high-performance, observability data pipeline for logs and metrics

Used in module log-shipper

Mozilla Public License, version 2.0

Используемые директории

Если узлы кластера Kubernetes анализируются сканерами безопасности (антивирусными средствами), то может потребоваться их настройка для исключения ложноположительных срабатываний.

Deckhouse Kubernetes Platform (DKP) использует следующие директории при работе (скачать в csv):

- `/mnt/kubernetes-data` (master-узел) — существует только в кластерах, развернутых в облаке, когда используется отдельный диск для базы данных etcd.
- `/var/lib/etcd` (master-узел) — база данных etcd.
- `/var/lib/deckhouse/` (master-узел) — файлы модулей Deckhouse, динамически загружаемых из хранилища образов.
- `/var/lib/upmeter` (master-узел, модуль `upmeter`) — база данных модуля upmeter.
- `/etc/kubernetes` (любой узел) — манифесты статических подов, файлы сертификатов PKI.
- `/var/lib/bashible` (любой узел) — файлы настройки узла.
- `/var/lib/containerd` (любой узел) — используется для хранения данных, связанных с работой CRI (например, containerd). Содержит слои образов контейнеров, снапшоты файловых систем контейнеров, метаданные, логи и другую информацию контейнеров.
- `/mnt/vector-data` (любой узел, модуль `log-shipper`) — служебные данные статуса отправленных журналов.
- `/var/log/containers` (любой узел) — журналы контейнеров (при использовании containerd).
- `/var/lib/kubelet/` (любой узел) — файлы настройки kubelet.
- `/opt/cni/bin/` (любой узел) — исполняемые файлы модуля CNI.
- `/opt/deckhouse/bin/` (любой узел) — исполняемые файлы, необходимые для работы Deckhouse.
- `/var/log/pods/` (любой узел) — журналы всех контейнеров подов, запущенных на узле.
- `/etc/cni/` (любой узел) — файлы настройки модуля CNI.

Описание и установка Deckhouse CLI

Deckhouse CLI (`d8`) — интерфейс командной строки для работы с продуктами экосистемы Deckhouse. Он объединяет команды для работы с Kubernetes, администрирования Deckhouse Kubernetes Platform, доставки приложений, копирования образов, резервного копирования, сбора диагностической информации, виртуализации, работы с пользовательскими данными и других задач.

Начиная с релиза 1.59, `d8` автоматически устанавливается на все узлы кластера DKP. Утилиту также можно установить на отдельную машину администратора.

Основные группы команд

i Группы команд `d8 delivery-kit` и `d8 mirror` недоступны для редакций Community Edition (CE) и Basic Edition (BE).

Команды `d8` сгруппированы по назначению:

Команды	Назначение
<code>d8 k</code>	Команды для управления кластером (аналогично <code>kubectl</code>).
<code>d8 system</code>	Команды для работы с системными настройками платформы, её конфигурацией, модулями, очередями, пакетами и системными логами.
<code>d8 status</code>	Команда для получения текущего состояния платформы.
<code>d8 backup</code>	Команды для создания резервных копий ключевых данных кластера и диагностических выгрузок.
<code>d8 cr</code>	Команды для работы с хранилищем образов контейнеров.
<code>d8 mirror</code>	Команды для копирования образов компонентов платформы в локальную файловую систему или стороннее хранилище образов контейнеров.
<code>d8 delivery-kit</code>	Команды для сборки, публикации, планирования и развёртывания приложений, а также для работы с Helm, хранилищем образов контейнеров, SBOM и связанными операциями.
<code>d8 data</code>	Команды для управления данными, а также их экспорта и импорта.
<code>d8 v</code>	Команды для работы с виртуальными машинами, созданными в кластере.
<code>d8 iam</code>	Команды для управления пользователями Deckhouse, группами и правами доступа.
<code>d8 stronghold</code>	Команды для управления жизненным циклом секретов (модуль <code>stronghold</code>).
<code>d8 network</code>	Команды для выполнения сетевых операций в экосистеме Deckhouse, включая миграцию CNI.
<code>d8 completion</code>	Команды для генерации <code>shell completion</code> .
<code>d8 tools</code>	Вспомогательные утилиты.
<code>d8 help</code>	Встроенная справка.

Как установить Deckhouse CLI

Установить Deckhouse CLI возможно двумя способами:

- Начиная с версии 0.10 доступна установка с помощью `trdl`. Такой способ позволяет непрерывно получать свежие версии утилиты со всеми доработками и исправлениями.

i Обратите внимание, что для установки через `trdl` необходим доступ в Интернет к туф-репозиторию с утилитой. В кластере с закрытым окружением такой способ работать не будет!

- Скачав исполняемый файл и установив его вручную.

Установка с помощью `trdl`

Начиная с версии 0.10 Deckhouse CLI можно установить с помощью trdl.



Если у вас установлена версия ниже 0.10, то её необходимо предварительно удалить.

Если вам нужно установить одну из версий ниже 0.10, воспользуйтесь устаревшим способом установки.

1. Установите клиент trdl.
2. Добавьте репозиторий Deckhouse CLI в trdl:

```
URL=https://deckhouse.ru/downloads/deckhouse-cli-trdl
ROOT_VERSION=1
ROOT_SHA512=343bd5f0d8811254e5f0b6fe292372a7b7eda08d276ff255229200f84e58a8151ab2729df3515cb11372dc3899c70df172a4e54c8a596a73d67ae790466a0491
REPO=d8

trdl add $REPO $URL $ROOT_VERSION $ROOT_SHA512
```

3. Установите последний стабильный выпуск утилиты `d8` и проверьте ее работоспособность:

```
.(trdl use d8 0 stable) && d8 --version
```

Если вы не хотите вызывать `.(trdl use d8 0 stable)` перед каждым использованием Deckhouse CLI, добавьте строку `alias d8='trdl exec d8 0 stable -- "$@"'` в RC-файл вашей командной оболочки.

Готово, вы установили Deckhouse CLI.

Установка исполняемого файла

- Linux x86-64
- macOS x86-64
- macOS ARM64
- Windows



Команды ниже актуальны для Deckhouse CLI версии v0.32.6, поставляемой с Deckhouse Kubernetes Platform. Последняя версия Deckhouse CLI доступна на GitHub. Узнать используемую версию Deckhouse CLI можно при помощи команды `d8 --version`.

1. Скачайте архив для Linux x86-64 удобным для вас способом:

- **Из вашего кластера:**

1. Включите модуль `deckhouse-tools`;
2. Откройте веб-интерфейс со ссылками на скачивание. Адрес веб-интерфейса формируется в соответствии с шаблоном `publicDomainTemplate` глобального параметра конфигурации Deckhouse (ключ `%s` заменяется на `tools`).

Например, если `publicDomainTemplate` установлен как `%s-kube.company.my`, веб-интерфейс будет доступен по адресу `tools-kube.company.my`;

3. Кликните по кнопке скачивания Deckhouse CLI для Linux.

- **С сайта Deckhouse:**

```
curl -LO https://deckhouse.ru/downloads/deckhouse-cli/v0.32.6/d8-v0.32.6-linux-amd64.tar.gz
```

2. Распакуйте архив:

```
tar -xvf "d8-v0.32.6-linux-amd64.tar.gz" "linux-amd64/bin/d8"
```

3. Переместите файл `d8` в каталог, находящийся в списке в переменной окружения `PATH` операционной системы (например, `/usr/local/bin/`):

```
sudo mv "linux-amd64/bin/d8" /usr/local/bin/
```

i Переменная окружения `PATH` содержит список каталогов, в которых ОС будет искать исполняемый файл при вызове команды из терминала. Просмотреть ее можно командой `echo $PATH`.

4. Проверьте, что утилита работает:

```
d8 help
```

Готово, вы установили Deckhouse CLI.

 Команды ниже актуальны для Deckhouse CLI версии v0.32.6, поставляемой с Deckhouse Kubernetes Platform. Последняя версия Deckhouse CLI доступна на GitHub. Узнать используемую версию Deckhouse CLI можно при помощи команды `d8 --version`.

1. Скачайте архив для macOS x86-64 удобным для вас способом:

◦ Из вашего кластера:

1. Включите модуль `deckhouse-tools`;
2. Откройте веб-интерфейс со ссылками на скачивание. Адрес веб-интерфейса формируется в соответствии с шаблоном `publicDomainTemplate` глобального параметра конфигурации Deckhouse (ключ `%s` заменяется на `tools`).

Например, если `publicDomainTemplate` установлен как `%s-kube.company.my`, веб-интерфейс будет доступен по адресу `tools-kube.company.my`;
3. Кликните по кнопке скачивания Deckhouse CLI для macOS.

◦ С сайта Deckhouse:

```
curl -LO https://deckhouse.ru/downloads/deckhouse-cli/v0.32.6/d8-v0.32.6-darwin-amd64.tar.gz
```

2. Распакуйте архив:

```
tar -xvf "d8-v0.32.6-darwin-amd64.tar.gz" "darwin-amd64/bin/d8"
```

3. Переместите файл `d8` в каталог, находящийся в списке в переменной окружения `PATH` операционной системы (например, `/usr/local/bin/`):

```
sudo mv "darwin-amd64/bin/d8" /usr/local/bin/
```

i Переменная окружения `PATH` содержит список каталогов, в которых ОС будет искать исполняемый файл при вызове команды из терминала. Просмотреть ее можно командой `echo $PATH`.

4. Проверьте, что утилита работает:

```
d8 help
```

Готово, вы установили Deckhouse CLI.

 Команды ниже актуальны для Deckhouse CLI версии v0.32.6, поставляемой с Deckhouse Kubernetes Platform. Последняя версия Deckhouse CLI доступна на GitHub. Узнать используемую версию Deckhouse CLI можно при помощи команды `d8 --version`.

1. Скачайте архив для macOS ARM64 удобным для вас способом:

- **Из вашего кластера:**

1. Включите модуль `deckhouse-tools`;
2. Откройте веб-интерфейс со ссылками на скачивание. Адрес веб-интерфейса формируется в соответствии с шаблоном `publicDomainTemplate` глобального параметра конфигурации Deckhouse (ключ `%s` заменяется на `tools`).

Например, если `publicDomainTemplate` установлен как `%s-kube.company.my`, веб-интерфейс будет доступен по адресу `tools-kube.company.my`;
3. Кликните по кнопке скачивания Deckhouse CLI для macOS.

- **С сайта Deckhouse:**

```
curl -LO https://deckhouse.ru/downloads/deckhouse-cli/v0.32.6/d8-v0.32.6-darwin-arm64.tar.gz
```

2. Распакуйте архив:

```
tar -xvf "d8-v0.32.6-darwin-arm64.tar.gz" "darwin-arm64/bin/d8"
```

3. Переместите файл `d8` в каталог, находящийся в списке в переменной окружения `PATH` операционной системы (например, `/usr/local/bin/`):

```
sudo mv "darwin-arm64/bin/d8" /usr/local/bin/
```

i Переменная окружения `PATH` содержит список каталогов, в которых ОС будет искать исполняемый файл при вызове команды из терминала. Просмотреть ее можно командой `echo $PATH`.

4. Проверьте, что утилита работает:

```
d8 help
```

Готово, вы установили Deckhouse CLI.

 Команды ниже актуальны для Deckhouse CLI версии v0.32.6, поставляемой с Deckhouse Kubernetes Platform. Последняя версия Deckhouse CLI доступна на GitHub. Узнать используемую версию Deckhouse CLI можно при помощи команды `d8 --version`.

1. Скачайте архив для Windows x86-64 удобным для вас способом.

- **Из вашего кластера:**

1. Включите модуль `deckhouse-tools`;
2. Откройте веб-интерфейс со ссылками на скачивание. Адрес веб-интерфейса формируется в соответствии с шаблоном `publicDomainTemplate` глобального параметра конфигурации Deckhouse (ключ `%s` заменяется на `tools`).

Например, если `publicDomainTemplate` установлен как `%s-kube.company.my`, веб-интерфейс будет доступен по адресу `tools-kube.company.my`;
3. Кликните по кнопке скачивания Deckhouse CLI для Windows.

- **С сайта Deckhouse**, по ссылке: Windows x86-64.

2. Распакуйте архив, переместите файл `d8.exe` в выбранный вами каталог и добавьте каталог в переменную `PATH` операционной системы.
3. Разблокируйте файл `d8.exe`, например, следующим способом:
 - Щелкните правой кнопкой мыши на файле и выберите *Свойства* в контекстном меню.
 - В окне *Свойства* убедитесь, что находитесь на вкладке *Общие*.
 - Внизу вкладки *Общие* вы можете увидеть раздел *Безопасность* с сообщением о блокировке файла.
 - Установите флажок *Разблокировать* или нажмите кнопку *Разблокировать*, затем нажмите *Применить* и *ОК*, чтобы сохранить изменения.
4. Проверьте, что утилита работает:

```
d8 help
```

5. Включите автодополнение в PowerShell, выполнив следующую команду:

```
d8 completion powershell >> $PROFILE
```

Готово, вы установили Deckhouse CLI.

Справка по командам Deckhouse CLI



Эта страница генерируется автоматически, информация представлена на английском языке.

d8 controls the Deckhouse Kubernetes Platform

Version: v0.32.6

Common options:

- `--help`, `-h`
help for d8
- `--version`, `-v`
version for d8

d8 backup

Backup various parts of Deckhouse Kubernetes Platform

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 backup cluster-config

Usage:

```
d8 backup cluster-config <backup-tarball-path> [options]
```

Take a snapshot of cluster configuration

Options

- `--compress`, `-c`
Compress result file

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 backup etcd

Usage:

```
d8 backup etcd <snapshot-path> [options]
```

Take a snapshot of ETCD state

Options

- `--etcd-pod`, `-p`
Name of the etcd pod to snapshot from. (optional)
- `--verbose`
Verbose log output.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 backup loki

Usage:

```
d8 backup loki [options]
```

Dump logs from Loki api.

Options

- `--days`
Limit maximum number of days in range to output per queue from Loki. (Default 5 maximum number of days in range per queue.)
- `--end`
Set end timestamp range to dump logs from Loki. (Default get max end timestamp from Loki, ex. "2025-01-14 15:04:05".)
- `--limit`
Limit the number of lines to output per queue from Loki. (Default 5000 max limit strings per queue.)
- `--start`
Set start timestamp range to dump logs from Loki. (Default get max start timestamp from Loki, ex. "2025-01-12 15:04:05".)

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 cli

Manage the deckhouse-cli (d8) binary

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--rpp-ca-file`
Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to \$D8_RPP_CA_FILE.
- `--rpp-endpoint`
registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to \$D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`
Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli check

Usage:

```
d8 cli check [options]
```

Report whether a newer deckhouse-cli version is available

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--rpp-ca-file`
Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to \$D8_RPP_CA_FILE.
- `--rpp-endpoint`
registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to \$D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`
Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli update

Usage:

```
d8 cli update [options]
```

Update deckhouse-cli to the latest version

Options

- `--version`
Exact version to install; downgrades are allowed (default: the latest).

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--rpp-ca-file`
Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to \$D8_RPP_CA_FILE.
- `--rpp-endpoint`
registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to \$D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`
Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli use

Usage:

```
d8 cli use <version> [options]
```

Switch to a specific deckhouse-cli version (no download when it is installed locally)

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--rpp-ca-file`
Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to \$D8_RPP_CA_FILE.
- `--rpp-endpoint`

registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to `$D8_RPP_ENDPOINT`.

- `--rpp-insecure-skip-tls-verify`

Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli versions

Alias: `list`

Usage:

```
d8 cli versions [options]
```

List deckhouse-cli versions available in the cluster registry

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)
- `--rpp-ca-file`
Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to `$D8_RPP_CA_FILE`.
- `--rpp-endpoint`
registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to `$D8_RPP_ENDPOINT`.
- `--rpp-insecure-skip-tls-verify`
Skip registry-packages-proxy TLS verification. For debugging only.

d8 completion

Generate the autocompletion script for the specified shell

d8 completion bash

Usage:

```
d8 completion bash [options]
```

Generate the autocompletion script for bash

Options

- `--no-descriptions`
disable completion descriptions

d8 completion fish

Usage:

```
d8 completion fish [options]
```

Generate the autocompletion script for fish

Options

- `--no-descriptions`
disable completion descriptions

d8 completion powershell

Usage:

```
d8 completion powershell [options]
```

Generate the autocompletion script for powershell

Options

- `--no-descriptions`
disable completion descriptions

d8 completion zsh

Usage:

```
d8 completion zsh [options]
```

Generate the autocompletion script for zsh

Options

- `--no-descriptions`
disable completion descriptions

d8 cr

Work with container registries

Options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with --username)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr catalog

Usage:

```
d8 cr catalog REGISTRY [options]
```

List the repositories in a registry

Options

- `--full-ref`
Print the full repository reference (registry/repo)

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with --username)
- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr config

Usage:

```
d8 cr config IMAGE [options]
```

Print the config of an image

Global options

- `--allow-nondistributable-artifacts`

Include non-distributable (foreign) layers when pushing

- `--insecure`

Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)

- `--password, -p`

Registry password or token (overrides the Docker config; use with `--username`)

- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr digest

Usage:

```
d8 cr digest [IMAGE] [options]
```

Print the digest of an image

Options

- `--full-ref`

Print the full image reference with digest (registry/repo@sha256:...); incompatible with `--tarball`

- `--tarball`

Read the digest from a local tarball instead of the registry

Global options

- `--allow-nondistributable-artifacts`

Include non-distributable (foreign) layers when pushing

- `--insecure`

Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)

- `--password, -p`

Registry password or token (overrides the Docker config; use with `--username`)

- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr export

Usage:

```
d8 cr export IMAGE [TARBALL] [options]
```

Export the filesystem of an image as a tarball

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform `os/arch[/variant][:osversion]` (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr fs**Usage:**

```
d8 cr fs [options]
```

Inspect or extract files inside a container image

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform `os/arch[/variant][:osversion]` (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr fs cat**Usage:**

```
d8 cr fs cat IMAGE PATH [options]
```

Print a file from a container image

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr fs extract

Usage:

```
d8 cr fs extract IMAGE [options]
```

Extract a container image filesystem to a local directory

Options

- `--output, -o`

Write the filesystem into this directory

Global options

- `--allow-nondistributable-artifacts`

Include non-distributable (foreign) layers when pushing

- `--insecure`

Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)

- `--password, -p`

Registry password or token (overrides the Docker config; use with `--username`)

- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr fs ls

Usage:

```
d8 cr fs ls IMAGE [PATH] [options]
```

List files inside a container image

Options

- `--long, -l`

Long format with mode and size

Global options

- `--allow-nondistributable-artifacts`

Include non-distributable (foreign) layers when pushing

- `--insecure`

Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)

- `--password, -p`

Registry password or token (overrides the Docker config; use with `--username`)

- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)

- `--username, -u`

Registry username (overrides the Docker config; use with `--password`)

- `--verbose, -v`

Enable debug logs on stderr

d8 cr fs tree

Usage:

```
d8 cr fs tree IMAGE [PATH] [options]
```

Show the filesystem of a container image as a tree

Options

- `--depth`, `-L`
Max depth to descend (0 = unlimited)
- `--dirsfirst`
List directories before files
- `--size`
Show file sizes (human-readable: B / KB / MB)

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose`, `-v`
Enable debug logs on stderr

d8 cr login

Usage:

```
d8 cr login [REGISTRY] [options]
```

Log in to a container registry

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose`, `-v`
Enable debug logs on stderr

d8 cr ls

Usage:

```
d8 cr ls REPO [options]
```

List the tags in a repository

Options

- `--full-ref`
Print the full image reference (registry/repo:tag)
- `--omit-digest-tags, -0`
Skip digest-based tags (sha256-*) created by signing tools

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with --username)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr manifest

Usage:

```
d8 cr manifest IMAGE [options]
```

Print the manifest of an image

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with --username)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with --password)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr pull

Usage:

```
d8 cr pull IMAGE... PATH [options]
```

Pull one or more remote images to a local path

Options

- `--cache-path, -c`
Cache image layers under this directory (reused between pulls)
- `--format`
Output format (one of: tarball, legacy, oci)

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose, -v`
Enable debug logs on stderr

d8 cr push**Usage:**

```
d8 cr push PATH IMAGE [options]
```

Push a local image to a registry

Options

- `--image-refs`
Persist the pushed reference (with digest) to this file
- `--index`
Push a multi-manifest OCI layout as an index (OCI layout dirs only)

Global options

- `--allow-nondistributable-artifacts`
Include non-distributable (foreign) layers when pushing
- `--insecure`
Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password, -p`
Registry password or token (overrides the Docker config; use with `--username`)
- `--platform`
Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username, -u`
Registry username (overrides the Docker config; use with `--password`)
- `--verbose, -v`
Enable debug logs on stderr

d8 data

Data operations (export/import)

d8 data create**Usage:**

```
d8 data create [flags] data_export_name volume_type/volume_name [options]
```

Deprecated: use 'd8 data export create'

Options

- `--namespace, -n`
data volume namespace

- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live

d8 data delete

Usage:

```
d8 data delete [flags] data_export_name [options]
```

Deprecated: use 'd8 data export delete'

Options

- `--namespace, -n`
data volume namespace

d8 data download

Usage:

```
d8 data download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Deprecated: use 'd8 data export download'

Options

- `--namespace, -n`
data volume namespace
- `--output, -o`
file to save data (default: same as resource)
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live for auto-created DataExport

d8 data export

Usage:

```
d8 data export [options]
```

Export data (DataExport)

d8 data export create

Usage:

```
d8 data export create [flags] data_export_name volume_type/volume_name [options]
```

Create dataexport kubernetes resource

Options

- `--namespace, -n`
data volume namespace
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live

d8 data export delete

Usage:

```
d8 data export delete [flags] data_export_name [options]
```

Delete dataexport kubernetes resource

Options

- `--namespace, -n`
data volume namespace

d8 data export download**Usage:**

```
d8 data export download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Download exported data

Options

- `--cleanup`
Delete auto-created DataExport without prompting (`--cleanup=true` to delete, `--cleanup=false` to keep)
- `--namespace, -n`
data volume namespace
- `--output, -o`
file to save data (default: same as resource)
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live for auto-created DataExport

d8 data export list

Alias: `ls`

Usage:

```
d8 data export list [flags] data_export_name [/path/] [options]
```

List DataExported content information

Options

- `--cleanup`
Delete auto-created DataExport without prompting (`--cleanup=true` to delete, `--cleanup=false` to keep)
- `--namespace, -n`
data volume namespace
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live for auto-created DataExport

d8 data import**Usage:**

```
d8 data import [options]
```

Import data (DataImport)

d8 data import create

Usage:

```
d8 data import create [flags] data_import_name [options]
```

Create DataImport

Options

- `--file, -f`
PVC manifest file path
- `--namespace, -n`
data volume namespace
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live
- `--wffc`
Wait for first consumer

d8 data import delete

Usage:

```
d8 data import delete [flags] data_import_name [options]
```

Delete dataimport kubernetes resource

Options

- `--namespace, -n`
data volume namespace

d8 data import upload

Usage:

```
d8 data import upload [flags] data_import_name path/file.ext [options]
```

Upload a file to the provided url

Options

- `--chunks, -c`
number of chunks to upload
- `--dstPath, -d`
destination path of the uploaded file
- `--file, -f`
file to upload
- `--namespace, -n`
data volume namespace
- `--publish, -P`
publish the uploaded file
- `--resume`
resume upload if process was interrupted

d8 data list

Usage:

```
d8 data list [flags] data_export_name [/path/] [options]
```

Deprecated: use 'd8 data export list'

Options

- `--namespace, -n`
data volume namespace
- `--publish`
Provide access outside of cluster
- `--ttl`
Time to live for auto-created DataExport

d8 delivery-kit

Alias: dk

A set of tools for building, distributing, and deploying containerized applications

d8 delivery-kit build

Usage:

```
d8 delivery-kit build [IMAGE_NAME...] [options]
```

Build images

Options

- `--add-custom-tag`
Set tag alias for the content-based tag.
The alias may contain the following shortcuts:
- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.
For cleaning custom tags and associated content-based tag are treated as one.
Also can be defined with \$WERF_ADD_CUSTOM_TAG_* (e.g.
\$WERF_ADD_CUSTOM_TAG_1="%image%-tag1", \$WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache

- (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--check-built-images`
Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_CHECK_BUILT_IMAGES`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or .werf in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`,

- `$WERF_DEV_IGNORE_DOCS=path/to/docs)`

 - `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default `$WERF_DIR` or current working directory)
 - `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like `werf-build`, `werf-converge` and other (default disabled or `WERF_DISABLE_AUTO_HOST_CLEANUP`)
 - `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
 - `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_DOCKER_SERVER_STORAGE_PATH`)
 - `--elf-pgp-private-key-base64`
Base64-encoded PGP private key (default `$WERF_ELF_PGP_PRIVATE_KEY_BASE64`)
 - `--elf-pgp-private-key-fingerprint`
PGP private key fingerprint (default `$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT`)
 - `--elf-pgp-private-key-passphrase`
Passphrase for the PGP private key (default `$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE`)
 - `--env`
Use specified environment (default `$WERF_ENV`)
 - `--final-images-only`
Process final images only (`$WERF_FINAL_IMAGES_ONLY` or false by default)
 - `--final-repo`
Container registry storage address (default `$WERF_FINAL_REPO`)
 - `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
 - `--final-repo-docker-hub-password`
final-repo Docker Hub password (default `$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD`)
 - `--final-repo-docker-hub-token`
final-repo Docker Hub token (default `$WERF_FINAL_REPO_DOCKER_HUB_TOKEN`)
 - `--final-repo-docker-hub-username`
final-repo Docker Hub username (default `$WERF_FINAL_REPO_DOCKER_HUB_USERNAME`)
 - `--final-repo-github-token`
final-repo GitHub token (default `$WERF_FINAL_REPO_GITHUB_TOKEN`)
 - `--final-repo-harbor-password`
final-repo Harbor password (default `$WERF_FINAL_REPO_HARBOR_PASSWORD`)
 - `--final-repo-harbor-username`
final-repo Harbor username (default `$WERF_FINAL_REPO_HARBOR_USERNAME`)
 - `--final-repo-quay-token`
final-repo quay.io token (default `$WERF_FINAL_REPO_QUAY_TOKEN`)
 - `--follow`
Enable follow mode (default `$WERF_FOLLOW`).
The mode allows restarting the command on a new commit.
In development mode (`--dev`), werf restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--introspect-before-error`
Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`
Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`
Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact
IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name
STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:

- * `$WERF_LOG_TERMINAL_WIDTH`
- * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default `$WERF_PARALLEL` or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default `$WERF_PARALLEL_TASKS_LIMIT` or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--project-name, -N`
Set a specific project name (default `$WERF_PROJECT_NAME`)
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)
- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--require-built-images, -Z`
Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions
- `--save-build-report`
Save build report (by default `$WERF_SAVE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)

- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_CERT`)
- `--sign-elf-files`
Enable ELF files signing (default `$WERF_SIGN_ELF_FILES`).
When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_INTERMEDIATES`)
- `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default `$WERF_SIGN_KEY`)
- `--sign-manifest`
Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).
When enabled,
the private signing key must be specified with `--sign-key` option and
the certificate must be specified with `--sign-cert` option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`,
`$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).
Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).
- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state
(`$WERF_VIRTUAL_MERGE` by default)

d8 delivery-kit bundle

Usage:

```
d8 delivery-kit bundle [options]
```

Work with werf bundles: publish bundles into container registry and deploy bundles into Kubernetes cluster

d8 delivery-kit bundle apply

Usage:

```
d8 delivery-kit bundle apply [options]
```

Apply bundle into Kubernetes

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g. \$WERF_ADD_ANNOTATION_1=annoName1=annoValue1, \$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`
Add label to deploying resources (can specify multiple).
Format: labelName=labelValue.
Also, can be specified with \$WERF_ADD_LABEL_* (e.g. \$WERF_ADD_LABEL_1=labelName1=labelValue1, \$WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`
Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed (\$WERF_ATOMIC by default)
- `--auto-rollback, -R`
Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed (\$WERF_AUTO_ROLLBACK by default)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`
Set default delete propagation strategy (default \$WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`
Path to the Deno binary to use instead of auto-downloading (default \$WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`
Save deploy graph path to the specified file (by default \$WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`
Change deploy report path and format (by default \$WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--disable-default-secret-values`
Do not use secret values from the default .helm/secret-values.yaml file (default \$WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`
Do not use values from the default .helm/values.yaml file (default \$WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`
Use specified environment (default \$WERF_ENV)

- `--force-adoption`
Always adopt resources, even if they belong to a different Helm release (default \$WERF_FORCE_ADOPTION or false)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`
No-op
- `--ignore-bundle-js`
Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default \$WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`
Disable secrets decryption (default \$WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-api-server`
Kubernetes API server address (default \$WERF_KUBE_API_SERVER)
- `--kube-auth-password`
Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`
Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`
Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`
Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`
Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)
- `--kube-ca-path`
Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with

- `$WERF_KUBE_IMPERSONATE_GROUP_*` environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_UID`)
- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_USER`)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY`)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY_DATA`)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default `$WERF_KUBE_PROXY_URL`)
- `--kube-qps-limit`
Kubernetes client QPS limit (default `$WERF_KUBE_QPS_LIMIT` or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default `$WERF_KUBE_REQUEST_TIMEOUT`)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default `$WERF_KUBE_TLS_SERVER`)
- `--kube-token`
Kubernetes bearer token used for authentication (default `$WERF_KUBE_TOKEN`)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default `$WERF_KUBE_TOKEN_PATH`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * `$WERF_LOG_TERMINAL_WIDTH`
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--namespace`
Use specified Kubernetes namespace (default `$WERF_NAMESPACE`)
- `--network-parallelism`
Parallelize some network operations (default `$WERF_NETWORK_PARALLELISM` or 30)

- `--no-final-tracking`
By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default \$WERF_NO_FINAL_TRACKING)
- `--no-install-crds`
Do not install CRDs from "crds/" directories of installed charts (default \$WERF_NO_INSTALL_CRDS)
- `--no-notes`
Don't show release notes at the end of the release (default \$WERF_NO_NOTES)
- `--no-pod-logs`
Disable Pod logs collection and printing (default \$WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default \$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
Strategy for provenance verifying (default \$WERF_PROVENANCE_STRATEGY).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default \$WERF_RELEASE)
- `--release-info-annotations`
Add annotations to release metadata (default \$WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`
Add Helm release labels (can specify multiple). Kind of labels depends on release storage driver. Format: labelName=labelValue.
Also, can be specified with \$WERF_RELEASE_LABEL_* (e.g. \$WERF_RELEASE_LABEL_1=labelName1=labelValue1, \$WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`
Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--render-subchart-notes`
If set, render subchart notes along with the parent (by default \$WERF_RENDER_SUBCHART_NOTES or false)
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

- repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--rollback-graph-path`
Save rollback graph path to the specified file (by default \$WERF_ROLLBACK_GRAPH_PATH).
Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default \$WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default \$WERF_RUNTIME_LABELS)
- `--save-deploy-report`
Save deploy report (by default \$WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`
Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1, \$WERF_SET_FILE_2=key2=val2)
- `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_JSON_* (e.g. \$WERF_SET_JSON_1=key1=val1, \$WERF_SET_JSON_2=key2=val2)
- `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_LITERAL_* (e.g. \$WERF_SET_LITERAL_1=key1=val1, \$WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`
Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead,

unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.

Also, can be defined with \$WERF_SET_ROOT_JSON_* (e.g. \$WERF_SET_ROOT_JSON_1=key1=val1, \$WERF_SET_ROOT_JSON_2=key2=val2)

- `--set-runtime-json`

Set new keys in \$.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.

Can specify multiple or separate values with commas: key1=val1,key2=val2.

Also, can be defined with \$WERF_SET_RUNTIME_JSON_* (e.g.

\$WERF_SET_RUNTIME_JSON_1=key1=val1, \$WERF_SET_RUNTIME_JSON_2=key2=val2)

- `--set-string`

Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).

Also, can be defined with \$WERF_SET_STRING_* (e.g. \$WERF_SET_STRING_1=key1=val1, \$WERF_SET_STRING_2=key2=val2)

- `--skip-dependencies-repo-refresh, -L`

Do not refresh helm chart repositories locally cached index

- `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)

- `--skip-tls-verify-kube`

Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--tag`

Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the specified bundle (\$WERF_TAG or latest by default)

- `--templates-allow-dns`

Allow performing DNS requests in templating (default \$WERF_TEMPLATES_ALLOW_DNS)

- `--timeout, -t`

Resources tracking timeout in seconds (\$WERF_TIMEOUT by default)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with \$WERF_VALUES_* (e.g. \$WERF_VALUES_1=.helm/values_1.yaml, \$WERF_VALUES_2=.helm/values_2.yaml)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit bundle copy

Usage:

```
d8 delivery-kit bundle copy [options]
```

Copy published bundle into another location

Options

- `--container-registry-mirror`

- (Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repos
- `--from`
Source address of the bundle to copy, specify bundle archive using schema `archive:PATH\_TO\_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.
- `--helm-compatible-chart, -C`
Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info <https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370>). In helm compatibility mode chart is fully conforming with the helm OCI registry requirements. Default true or \$WERF_HELM_COMPATIBLE_CHART.
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes. Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--rename-chart`
Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be

set by the \$WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--to`

Destination address of the bundle to copy, specify bundle archive using schema

`archive:PATH\_TO\_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit bundle plan

Usage:

```
d8 delivery-kit bundle plan [options]
```

Plan bundle apply into Kubernetes

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: annoName=annoValue.

Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.

\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.

Also, can be specified with \$WERF_ADD_LABEL_* (e.g.

\$WERF_ADD_LABEL_1=labelName1=labelValue1,

\$WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

- `--delete-propagation`

Set default delete propagation strategy (default \$WERF_DELETE_PROPAGATION or Foreground).

- `--deno-binary-path`

Path to the Deno binary to use instead of auto-downloading (default \$WERF_DENO_BINARY_PATH)

- `--deploy-graph-path`

Save deploy graph path to the specified file (by default \$WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used

- `--diff-context-lines`

Show N lines of context around diffs (\$WERF_DIFF_CONTEXT_LINES by default)

- `--disable-default-secret-values`

Do not use secret values from the default .helm/secret-values.yaml file (default \$WERF_DISABLE_DEFAULT_SECRET_VALUES or false)

- `--disable-default-values`

Do not use values from the default .helm/values.yaml file (default

- `$WERF_DISABLE_DEFAULT_VALUES` or false)
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`
Use specified environment (default `$WERF_ENV`)
- `--exit-code`
Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default `$WERF_EXIT_CODE` or false)
- `--force-adoption`
Always adopt resources, even if they belong to a different Helm release (default `$WERF_FORCE_ADOPTION` or false)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--ignore-bundle-js`
Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default `$WERF_IGNORE_BUNDLE_JS`)
- `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default `$WERF_INSECURE_HELM_DEPENDENCIES`)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default `$WERF_INSECURE_REGISTRY`)
- `--kube-api-server`
Kubernetes API server address (default `$WERF_KUBE_API_SERVER`)
- `--kube-auth-password`
Basic auth password for Kubernetes API (default `$WERF_KUBE_AUTH_PASSWORD`)
- `--kube-auth-provider`
Auth provider name for authentication in Kubernetes API (default `$WERF_KUBE_AUTH_PROVIDER`)
- `--kube-auth-provider-config`
Auth provider config for authentication in Kubernetes API (default `$WERF_KUBE_AUTH_PROVIDER_CONFIG`)
- `--kube-auth-username`
Basic auth username for Kubernetes API (default `$WERF_KUBE_AUTH_USERNAME`)
- `--kube-burst-limit`
Kubernetes client burst limit (default `$WERF_KUBE_BURST_LIMIT` or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default `$WERF_KUBE_CA_DATA`)
- `--kube-ca-path`
Kubernetes API server CA path (default `$WERF_KUBE_CA_PATH`)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT`)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT_DATA`)
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or

- `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_CLUSTER`)
- `--kube-context-user`
Use user from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_USER`)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with `$WERF_KUBE_IMPERSONATE_GROUP_*` environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_UID`)
- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_USER`)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY`)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY_DATA`)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default `$WERF_KUBE_PROXY_URL`)
- `--kube-qps-limit`
Kubernetes client QPS limit (default `$WERF_KUBE_QPS_LIMIT` or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default `$WERF_KUBE_REQUEST_TIMEOUT`)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default `$WERF_KUBE_TLS_SERVER`)
- `--kube-token`
Kubernetes bearer token used for authentication (default `$WERF_KUBE_TOKEN`)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default `$WERF_KUBE_TOKEN_PATH`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * `$WERF_LOG_TERMINAL_WIDTH`
 - * interactive terminal width or 140
- `--log-time`

- Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace`
Use specified Kubernetes namespace (default \$WERF_NAMESPACE)
- `--network-parallelism`
Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`
By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default \$WERF_NO_FINAL_TRACKING)
- `--no-install-crds`
Do not install CRDs from "crds/" directories of installed charts (default \$WERF_NO_INSTALL_CRDS)
- `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default \$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
Strategy for provenance verifying (default \$WERF_PROVENANCE_STRATEGY).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default \$WERF_RELEASE)
- `--release-info-annotations`
Add annotations to release metadata (default \$WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`
Add Helm release labels (can specify multiple). Kind of labels depends on release storage driver.
Format: `labelName=labelValue`.
Also, can be specified with `$WERF_RELEASE_LABEL_*` (e.g.
`$WERF_RELEASE_LABEL_1=labelName1=labelValue1`,
`$WERF_RELEASE_LABEL_2=labelName2=labelValue2`)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

- repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default \$WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default \$WERF_RUNTIME_LABELS)
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`
Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1, \$WERF_SET_FILE_2=key2=val2)
- `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_JSON_* (e.g. \$WERF_SET_JSON_1=key1=val1, \$WERF_SET_JSON_2=key2=val2)
- `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_LITERAL_* (e.g. \$WERF_SET_LITERAL_1=key1=val1, \$WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`
Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with \$WERF_SET_ROOT_JSON_* (e.g. \$WERF_SET_ROOT_JSON_1=key1=val1, \$WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`
Set new keys in \$.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.

- Can specify multiple or separate values with commas: key1=val1,key2=val2.
- Also, can be defined with \$WERF_SET_RUNTIME_JSON_* (e.g. \$WERF_SET_RUNTIME_JSON_1=key1=val1, \$WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`
Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_STRING_* (e.g. \$WERF_SET_STRING_1=key1=val1, \$WERF_SET_STRING_2=key2=val2)
- `--show-insignificant-diffs`
Show insignificant diff lines (\$WERF_SHOW_INSIGNIFICANT_DIFFS by default)
- `--show-sensitive-diffs`
Show sensitive diff lines (\$WERF_SHOW_SENSITIVE_DIFFS by default)
- `--show-verbose-crd-diffs`
Show verbose CRD diff lines (\$WERF_SHOW_VERBOSE_CRD_DIFFS by default)
- `--show-verbose-diffs`
Show verbose diff lines (\$WERF_SHOW_VERBOSE_DIFFS by default)
- `--skip-dependencies-repo-refresh, -L`
Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`
Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tag`
Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the specified bundle (\$WERF_TAG or latest by default)
- `--templates-allow-dns`
Allow performing DNS requests in templating (default \$WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--values`
Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with \$WERF_VALUES_* (e.g. \$WERF_VALUES_1=.helm/values_1.yaml, \$WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit bundle publish

Usage:

```
d8 delivery-kit bundle publish [IMAGE_NAME...] [options]
```

Publish bundle

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g. \$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

- `$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)`

 - `--add-custom-tag`
Set tag alias for the content-based tag.
The alias may contain the following shortcuts:
 - `%image%`, `%image_slug%` or `%image_safe_slug%` to use the image name (necessary if there is more than one image in the werf config);
 - `%image_content_based_tag%` to use a content-based tag.
 For cleaning custom tags and associated content-based tag are treated as one.
Also can be defined with `$WERF_ADD_CUSTOM_TAG_*` (e.g. `$WERF_ADD_CUSTOM_TAG_1="%image%-tag1"`, `$WERF_ADD_CUSTOM_TAG_2="%image%-tag2"`)
 - `--add-label`
Add label to deploying resources (can specify multiple).
Format: `labelName=labelValue`.
Also, can be specified with `$WERF_ADD_LABEL_*` (e.g. `$WERF_ADD_LABEL_1=labelName1=labelValue1`, `$WERF_ADD_LABEL_2=labelName2=labelValue2`)
 - `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
 - `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE`)
 - `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "`allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin`" level (default 5% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN`)
 - `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`)
 - `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "`allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin`" level (default 5% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`)
 - `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
 - `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "`allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin`" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
 - `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
 - `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
 - `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
 - `--bsign-elf-files`

Enable ELF files signing with bsign (default \$WERF_BSIGN_ELF_FILES).

When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option

- `--build-report-path`

Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

- `--deno-binary-path`

Path to the Deno binary to use instead of auto-downloading (default \$WERF_DENO_BINARY_PATH)

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)

- `--disable-default-secret-values`

Do not use secret values from the default .helm/secret-values.yaml file (default \$WERF_DISABLE_DEFAULT_SECRET_VALUES or false)

- `--disable-default-values`

Do not use values from the default .helm/values.yaml file (default \$WERF_DISABLE_DEFAULT_VALUES or false)

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

- Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
 - `--elf-gpg-private-key-base64`
Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
 - `--elf-gpg-private-key-fingerprint`
PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
 - `--elf-gpg-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
 - `--env`
Use specified environment (default \$WERF_ENV)
 - `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or true by default)
 - `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
 - `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
 - `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
 - `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
 - `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
 - `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
 - `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
 - `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
 - `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
 - `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
 - `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
 - `--helm-compatible-chart, -C`
Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info <https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370>).
In helm compatibility mode chart is fully conforming with the helm OCI registry requirements.
Default false or \$WERF_HELM_COMPATIBLE_CHART.
 - `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
 - `--ignore-secret-key`

Disable secrets decryption (default \$WERF_IGNORE_SECRET_KEY)

- `--insecure-helm-dependencies`

Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage

- `--introspect-error`

Introspect failed stage in the state, right after running failed assembly instruction

- `--introspect-stage`

Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`

Print current project directory path (default \$WERF_LOG_PROJECT_DIR)

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

- Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--rename-chart`
Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be set by the \$WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`
Save build report (by default \$WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`

- Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
 - `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1, \$WERF_SET_FILE_2=key2=val2)
 - `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_JSON_* (e.g. \$WERF_SET_JSON_1=key1=val1, \$WERF_SET_JSON_2=key2=val2)
 - `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_LITERAL_* (e.g. \$WERF_SET_LITERAL_1=key1=val1, \$WERF_SET_LITERAL_2=key2=val2)
 - `--set-root-json`
Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with \$WERF_SET_ROOT_JSON_* (e.g. \$WERF_SET_ROOT_JSON_1=key1=val1, \$WERF_SET_ROOT_JSON_2=key2=val2)
 - `--set-runtime-json`
Set new keys in \$.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with \$WERF_SET_RUNTIME_JSON_* (e.g. \$WERF_SET_RUNTIME_JSON_1=key1=val1, \$WERF_SET_RUNTIME_JSON_2=key2=val2)
 - `--set-string`
Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_STRING_* (e.g. \$WERF_SET_STRING_1=key1=val1, \$WERF_SET_STRING_2=key2=val2)
 - `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)
 - `--sign-elf-files`
Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).
When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
 - `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)
 - `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)

- `--sign-manifest`
Enable image manifest signing (default \$WERF_SIGN_MANIFEST).
When enabled,
the private signing key must be specified with `--sign-key` option and
the certificate must be specified with `--sign-cert` option
- `--skip-dependencies-repo-refresh, -L`
Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`
Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`
Publish bundle into container registry repo by the provided tag (\$WERF_TAG or latest by default)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--use-build-report`
Use build report, previously saved with `--save-build-report` (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with `--build-report-path`
- `--use-custom-tag`
Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
The alias may contain the following shortcuts:
- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.
For cleaning custom tags and associated content-based tag are treated as one.
Also, can be defined with \$WERF_USE_CUSTOM_TAG (e.g. \$WERF_USE_CUSTOM_TAG="%image%-tag")
- `--values`
Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with \$WERF_VALUES_* (e.g. \$WERF_VALUES_1=.helm/values_1.yaml, \$WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).
- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state

(\$WERF_VIRTUAL_MERGE by default)

- `--without-images`

Disable building of images defined in the `werf.yaml` (if any) and usage of such images in the `.helm/templates` (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the `werf.yaml` by default)

d8 delivery-kit bundle render

Usage:

```
d8 delivery-kit bundle render [options]
```

Render Kubernetes manifests from bundle

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: `annoName=annoValue`.

Also, can be specified with `$WERF_ADD_ANNOTATION_*` (e.g.

`$WERF_ADD_ANNOTATION_1=annoName1=annoValue1`,

`$WERF_ADD_ANNOTATION_2=annoName2=annoValue2`)

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: `labelName=labelValue`.

Also, can be specified with `$WERF_ADD_LABEL_*` (e.g.

`$WERF_ADD_LABEL_1=labelName1=labelValue1`,

`$WERF_ADD_LABEL_2=labelName2=labelValue2`)

- `--bundle-dir, -b`

Get extracted bundle from directory instead of registry (default `$WERF_BUNDLE_DIR`)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)

- `--deno-binary-path`

Path to the Deno binary to use instead of auto-downloading (default `$WERF_DENO_BINARY_PATH`)

- `--disable-default-secret-values`

Do not use secret values from the default `.helm/secret-values.yaml` file (default

`$WERF_DISABLE_DEFAULT_SECRET_VALUES` or false)

- `--disable-default-values`

Do not use values from the default `.helm/values.yaml` file (default

`$WERF_DISABLE_DEFAULT_VALUES` or false)

- `--docker-config`

Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or

`~/docker` (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo, to pull base images

- `--env`

Use specified environment (default `$WERF_ENV`)

- `--extra-apiversions`

Extra Kubernetes API versions passed to `$.Capabilities.APIVersions`. Can be also set with

`$WERF_EXTRA_APIVERSIONS_*` environment variables, values can be comma-separated

- `--force-adoption`

Always adopt resources, even if they belong to a different Helm release (default

- `$WERF_FORCE_ADOPTION` or false)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--ignore-bundle-js`
Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default `$WERF_IGNORE_BUNDLE_JS`)
- `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
- `--include-crds`
Include CRDs in the templated output (default `$WERF_INCLUDE_CRDS`)
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default `$WERF_INSECURE_HELM_DEPENDENCIES`)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default `$WERF_INSECURE_REGISTRY`)
- `--kube-api-server`
Kubernetes API server address (default `$WERF_KUBE_API_SERVER`)
- `--kube-auth-password`
Basic auth password for Kubernetes API (default `$WERF_KUBE_AUTH_PASSWORD`)
- `--kube-auth-provider`
Auth provider name for authentication in Kubernetes API (default `$WERF_KUBE_AUTH_PROVIDER`)
- `--kube-auth-provider-config`
Auth provider config for authentication in Kubernetes API (default `$WERF_KUBE_AUTH_PROVIDER_CONFIG`)
- `--kube-auth-username`
Basic auth username for Kubernetes API (default `$WERF_KUBE_AUTH_USERNAME`)
- `--kube-burst-limit`
Kubernetes client burst limit (default `$WERF_KUBE_BURST_LIMIT` or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default `$WERF_KUBE_CA_DATA`)
- `--kube-ca-path`
Kubernetes API server CA path (default `$WERF_KUBE_CA_PATH`)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT`)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT_DATA`)
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_CLUSTER`)
- `--kube-context-user`
Use user from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_USER`)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with `$WERF_KUBE_IMPERSONATE_GROUP_*` environment variables
- `--kube-impersonate-uid`

- Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`
- Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
- Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)
- `--kube-key-data`
- Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`
- Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`
- Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`
- Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`
- Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
- `--kube-token`
- Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
- `--kube-token-path`
- Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
- `--kube-version`
- Set specific Capabilities.KubeVersion (default \$WERF_KUBE_VERSION)
- `--log-color-mode`
- Set log color mode.
- Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
- Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
- Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
- Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
- Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
- Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
- Set log terminal width.
- Defaults to:
- `--log-time`
- * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
- Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
- Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace`
- Use specified Kubernetes namespace (default \$WERF_NAMESPACE)
- `--network-parallelism`
- Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)

- `--output`
Write render output to the specified file instead of stdout (\$WERF_RENDER_OUTPUT by default)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default \$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
Strategy for provenance verifying (default \$WERF_PROVENANCE_STRATEGY).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default \$WERF_RELEASE)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`
Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate

- values with commas: key1=path1,key2=path2).
- Also, can be defined with `$WERF_SET_FILE_*` (e.g. `$WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2`)
- `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with `$WERF_SET_JSON_*` (e.g. `$WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2`)
 - `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with `$WERF_SET_LITERAL_*` (e.g. `$WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2`)
 - `--set-root-json`
Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with `$WERF_SET_ROOT_JSON_*` (e.g. `$WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2`)
 - `--set-runtime-json`
Set new keys in `$.Runtime`, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with `$WERF_SET_RUNTIME_JSON_*` (e.g. `$WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2`)
 - `--set-string`
Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with `$WERF_SET_STRING_*` (e.g. `$WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2`)
 - `--show-only, -s`
only show manifests rendered from the given templates
 - `--skip-dependencies-repo-refresh, -L`
Do not refresh helm chart repositories locally cached index
 - `--skip-tls-verify-helm-dependencies`
Skip TLS certificate validation when accessing a Helm charts repository (default `$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES`)
 - `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default `$WERF_SKIP_TLS_VERIFY_KUBE`)
 - `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
 - `--tag`
Provide exact tag version or semver-based pattern, werf will render the latest version of the specified bundle (`$WERF_TAG` or latest by default)
 - `--templates-allow-dns`
Allow performing DNS requests in templating (default `$WERF_TEMPLATES_ALLOW_DNS`)
 - `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
 - `--validate`
Validate your manifests against the Kubernetes cluster you are currently pointing at (default

- `$WERF_VALIDATE)`
- `--values`
Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with `$WERF_VALUES_*` (e.g. `$WERF_VALUES_1=.helm/values_1.yaml`, `$WERF_VALUES_2=.helm/values_2.yaml`)
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit chart

Usage:

```
d8 delivery-kit chart [options]
```

Work with charts

d8 delivery-kit chart ts

Usage:

```
d8 delivery-kit chart ts [options]
```

Work with TypeScript charts

d8 delivery-kit chart ts build

Usage:

```
d8 delivery-kit chart ts build [PATH] [options]
```

Build TypeScript chart.

Options

- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--deno-binary-path`
Path to the Deno binary to use instead of auto-downloading (default `$WERF_DENO_BINARY_PATH`)
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--env`

- Use specified environment (default \$WERF_ENV)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit chart ts init

Usage:

```
d8 delivery-kit chart ts init [PATH] [options]
```

Initialize the files needed to render manifests using TypeScript.

Options

- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--env`
Use specified environment (default `$WERF_ENV`)
- `--git-work-tree`
Use specified git work tree dir (default `$WERF_WORK_TREE` or lookup for directory that contains `.git` in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default `$WERF_GITERMINISM_CONFIG` or `werf-giterminism.yaml` in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit ci-env

Usage:

```
d8 delivery-kit ci-env CI_SYSTEM [options]
```

Generate werf environment variables for specified CI system

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--as-env-file`
Create the .env file and print the path for sourcing (default \$WERF_AS_ENV_FILE).
- `--as-file`
Create the script and print the path for sourcing (default \$WERF_AS_FILE).
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command will copy specified or default (~/.docker) config to the temporary directory and may perform additional login with new config.
- `--env`
Use specified environment (default \$WERF_ENV)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

- Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--login-to-registry`
Log in to CI-specific registry automatically if possible (default \$WERF_LOGIN_TO_REGISTRY).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to custom file (default \$WERF_OUTPUT_FILE_PATH).
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--shell`
Set to cmdexe, powershell or use the default behavior that is compatible with any unix shell
(default \$WERF_SHELL).
- `--tagging-strategy`
stub
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--use-docker-auth-config`
Generate Docker config from DOCKER_AUTH_CONFIG environment variable instead of copying
current Docker config (default \$WERF_USE_DOCKER_AUTH_CONFIG).
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit cleanup

Usage:

```
d8 delivery-kit cleanup [options]
```

Cleanup project images in the container registry

Options

- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE`)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/.werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)
- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read, pull and delete images from the specified repo

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)

- `--dry-run`

Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)

- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

- Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
 - `--insecure-helm-dependencies`
No-op
 - `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
 - `--keep-list`
Set path to keep list file (default \$WERF_KEEP_LIST)
 - `--keep-stages-built-within-last-n-hours`
Keep stages that were built within last hours (default \$WERF_KEEP_STAGES_BUILT_WITHIN_LAST_N_HOURS or 2)
 - `--kube-api-server`
Kubernetes API server address (default \$WERF_KUBE_API_SERVER)
 - `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)
 - `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
 - `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
 - `--kube-context`
Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or \$WERF_SCAN_CONTEXT_ONLY)
 - `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
 - `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
 - `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
 - `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
 - `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
 - `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
 - `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
 - `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
 - `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
 - `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
 - `--log-verbose`

- Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--scan-context-namespace-only`
Scan for used images only in namespace linked with context for each available context in kube-config (or only for the context specified with option --kube-context). When disabled will scan all namespaces in all contexts (or only for the context specified with option --kube-context). (Default \$WERF_SCAN_CONTEXT_NAMESPACE_ONLY)
- `--scan-context-only`
Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or \$WERF_SCAN_CONTEXT_ONLY)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default

\$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--without-kube`

Do not skip deployed Kubernetes images (default \$WERF_WITHOUT_KUBE)

d8 delivery-kit completion

Usage:

```
d8 delivery-kit completion [options]
```

Generate bash completion scripts

Options

- `--shell`

Set to bash, zsh, fish or powershell (default \$WERF_SHELL or bash)

d8 delivery-kit compose

Usage:

```
d8 delivery-kit compose [options]
```

Work with docker-compose

d8 delivery-kit compose config

Usage:

```
d8 delivery-kit compose config [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"]
```

Run docker-compose config command with forwarded image names.

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--allowed-backend-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/.werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`

- Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

 - `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default \$WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

Define docker-compose command options (default \$WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

Define docker-compose options (default \$WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

Use specified environment (default \$WERF_ENV)
- `--final-images-only`

Process final images only (\$WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--project-name, -N`
Set a specific project name (default `$WERF_PROJECT_NAME`)
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)
- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_REQUIRE_BUILT_IMAGES`)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...`)
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_CERT`)
- `--sign-elf-files`
Enable ELF files signing (default `$WERF_SIGN_ELF_FILES`).
When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_INTERMEDIATES`)
- `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default

- `$WERF_SIGN_KEY`
- `--sign-manifest`
Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).
When enabled,
the private signing key must be specified with `--sign-key` option and
the certificate must be specified with `--sign-cert` option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`,
`$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).
Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`
- `--stub-tags`
Use stubs instead of real tags (default `$WERF_STUB_TAGS`)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--use-build-report`
Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).
- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state (`$WERF_VIRTUAL_MERGE` by default)
- `--without-images`
Disable building of images defined in the `werf.yaml` (if any) and usage of such images in the `.helm/templates` (`$WERF_WITHOUT_IMAGES` or false by default — e.g. enable all images defined in the `werf.yaml` by default)

d8 delivery-kit compose down

Usage:

```
d8 delivery-kit compose down [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"]
```

Run docker-compose down command with forwarded image names.

Options

- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--allowed-backend-storage-volume-usage`

- Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE`)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/.werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`

- Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
 - `--debug`
Enable debug (default \$WERF_DEBUG).
 - `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
 - `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
 - `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
 - `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
 - `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
 - `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
 - `--docker-compose-bin-path`
DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default \$WERF_DOCKER_COMPOSE_BIN_PATH)
 - `--docker-compose-command-options`
Define docker-compose command options (default \$WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
 - `--docker-compose-options`
Define docker-compose options (default \$WERF_DOCKER_COMPOSE_OPTIONS)
 - `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read and pull images from the specified repo
 - `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
 - `--dry-run`
Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
 - `--elf-pgp-private-key-base64`
Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
 - `--elf-pgp-private-key-fingerprint`
PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
 - `--elf-pgp-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
 - `--env`
Use specified environment (default \$WERF_ENV)
 - `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or false by default)
 - `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)

- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--insecure-helm-dependencies`

No-op

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`

Print current project directory path (default \$WERF_LOG_PROJECT_DIR)

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * `$WERF_LOG_TERMINAL_WIDTH`
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--project-name, -N`
Set a specific project name (default `$WERF_PROJECT_NAME`)
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)
- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_REQUIRE_BUILT_IMAGES`)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_CERT`)
- `--sign-elf-files`
Enable ELF files signing (default `$WERF_SIGN_ELF_FILES`).

When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option

- `--sign-intermediates`

The intermediate certificates as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_INTERMEDIATES`)

- `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default `$WERF_SIGN_KEY`)

- `--sign-manifest`

Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).

When enabled,

the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option

- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`, `$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).

Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`

- `--stub-tags`

Use stubs instead of real tags (default `$WERF_STUB_TAGS`)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)

- `--use-build-report`

Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--verbose`

Enable verbose output (default `$WERF_VERBOSE`).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (`$WERF_VIRTUAL_MERGE` by default)

- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (`$WERF_WITHOUT_IMAGES` or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose run

Usage:

```
d8 delivery-kit compose run [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] -- SERVICE [COMMAND] [ARGS...]
```

Run docker-compose run command with forwarded image names.

Options

- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE`)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/.werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `~/.werf-build-report.json` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing

images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)

- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)

- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)

- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)

- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like `werf-build`, `werf-converge` and other (default disabled or `WERF_DISABLE_AUTO_HOST_CLEANUP`)

- `--docker-compose-bin-path`

DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default `$WERF_DOCKER_COMPOSE_BIN_PATH`)

- `--docker-compose-command-options`

Define docker-compose command options (default `$WERF_DOCKER_COMPOSE_COMMAND_OPTIONS`)

- `--docker-compose-options`

Define docker-compose options (default `$WERF_DOCKER_COMPOSE_OPTIONS`)

- `--docker-config`

Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)

Command needs granted permissions to read and pull images from the specified repo

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_DOCKER_SERVER_STORAGE_PATH`)

- `--dry-run`

Indicate what the command would do without actually doing that (default `$WERF_DRY_RUN`)

- `--elf-pgp-private-key-base64`

Base64-encoded PGP private key (default `$WERF_ELF_PGP_PRIVATE_KEY_BASE64`)

- `--elf-pgp-private-key-fingerprint`

- PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)

- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_CERT`)
- `--sign-elf-files`
Enable ELF files signing (default `$WERF_SIGN_ELF_FILES`).
When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_INTERMEDIATES`)
- `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default `$WERF_SIGN_KEY`)
- `--sign-manifest`
Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).
When enabled,
the private signing key must be specified with `--sign-key` option and
the certificate must be specified with `--sign-cert` option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`, `$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).
Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`
- `--stub-tags`
Use stubs instead of real tags (default `$WERF_STUB_TAGS`)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--use-build-report`
Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).
- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state (`$WERF_VIRTUAL_MERGE` by default)
- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose up

Usage:

```
d8 delivery-kit compose up [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] [--] [SERVICE...]
```

Run docker-compose up command with forwarded image names.

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default \$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`
Network mode for the build containers (\$WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default \$WERF_BSIGN_ELF_FILES).
When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)

- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)

- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)

- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)

- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like `werf-build`, `werf-converge` and other (default disabled or `WERF_DISABLE_AUTO_HOST_CLEANUP`)

- `--docker-compose-bin-path`

DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default `$WERF_DOCKER_COMPOSE_BIN_PATH`)

- `--docker-compose-command-options`

Define docker-compose command options (default `$WERF_DOCKER_COMPOSE_COMMAND_OPTIONS`)

- `--docker-compose-options`

Define docker-compose options (default `$WERF_DOCKER_COMPOSE_OPTIONS`)

- `--docker-config`

Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)

Command needs granted permissions to read and pull images from the specified repo

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage

- volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`
 - Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`
 - Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`
 - PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`
 - Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
 - Use specified environment (default \$WERF_ENV)
- `--final-images-only`
 - Process final images only (\$WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`
 - Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
 - Choose final-repo container registry implementation.
 - The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
 - Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
 - final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
 - final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
 - final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
 - final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
 - final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
 - final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
 - final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`
 - Enable follow mode (default \$WERF_FOLLOW).
 - The mode allows restarting the command on a new commit.
 - In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`
 - Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
 - Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
 - Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
 - No-op
- `--insecure-registry`
 - Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

- repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)
- `--sign-elf-files`
Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).
When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)
- `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)
- `--sign-manifest`
Enable image manifest signing (default \$WERF_SIGN_MANIFEST).
When enabled,
the private signing key must be specified with --sign-key option and
the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`
Use stubs instead of real tags (default \$WERF_STUB_TAGS)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

Use build report, previously saved with --save-build-report (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit config

Usage:

```
d8 delivery-kit config [options]
```

Work with werf.yaml

d8 delivery-kit config graph

Usage:

```
d8 delivery-kit config graph [IMAGE_NAME...] [options]
```

Print dependency graph for images in werf.yaml.

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or false by default)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit config list

Usage:

```
d8 delivery-kit config list [options]
```

List image and artifact names defined in werf.yaml.

Options

- `--allow-includes-update`

- Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-images-only`
Show only final images
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--images-only`
Show only final images
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH

- * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit config render

Usage:

```
d8 delivery-kit config render [IMAGE_NAME...] [options]
```

Render werf.yaml.

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains

- .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit converge

Usage:

```
d8 delivery-kit converge [options]
```

Build and push images, then deploy application into Kubernetes

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.
\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-custom-tag`
Set tag alias for the content-based tag.
The alias may contain the following shortcuts:
- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is

more than one image in the werf config);

- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also can be defined with \$WERF_ADD_CUSTOM_TAG_* (e.g.

\$WERF_ADD_CUSTOM_TAG_1="%image%-tag1", \$WERF_ADD_CUSTOM_TAG_2="%image%-tag2")

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.

Also, can be specified with \$WERF_ADD_LABEL_* (e.g.

\$WERF_ADD_LABEL_1=labelName1=labelValue1,

\$WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--allowed-backend-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)

- `--allowed-docker-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-local-cache-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)

- `--allowed-local-cache-volume-usage-margin`

During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)

- `--annotate-layers-with-dm-verity-root-hash`

Enable annotation of image layers with dm-verity root hash (default

\$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)

- `--atomic`

Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed (\$WERF_ATOMIC by default)

- `--auto-rollback, -R`

Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed (\$WERF_AUTO_ROLLBACK by default)

- `--backend-network`

Network mode for the build containers (\$WERF_BACKEND_NETWORK or nothing by default)

- `--backend-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)

- `--bsign-elf-files`
Enable ELF files signing with bsign (default \$WERF_BSIGN_ELF_FILES).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`
Set default delete propagation strategy (default \$WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`
Path to the Deno binary to use instead of auto-downloading (default \$WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`
Save deploy graph path to the specified file (by default \$WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`
Change deploy report path and format (by default \$WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like `werf-build`, `werf-converge` and other (default disabled or `WERF_DISABLE_AUTO_HOST_CLEANUP`)

- `--disable-default-secret-values`

Do not use secret values from the default `.helm/secret-values.yaml` file (default `$WERF_DISABLE_DEFAULT_SECRET_VALUES` or `false`)

- `--disable-default-values`

Do not use values from the default `.helm/values.yaml` file (default `$WERF_DISABLE_DEFAULT_VALUES` or `false`)

- `--docker-config`

Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/docker` (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo, to pull base images

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_DOCKER_SERVER_STORAGE_PATH`)

- `--elf-gpg-private-key-base64`

Base64-encoded PGP private key (default `$WERF_ELF_PGP_PRIVATE_KEY_BASE64`)

- `--elf-gpg-private-key-fingerprint`

PGP private key fingerprint (default `$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT`)

- `--elf-gpg-private-key-passphrase`

Passphrase for the PGP private key (default `$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE`)

- `--env`

Use specified environment (default `$WERF_ENV`)

- `--final-images-only`

Process final images only (`$WERF_FINAL_IMAGES_ONLY` or `true` by default)

- `--final-repo`

Container registry storage address (default `$WERF_FINAL_REPO`)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: `ecr`, `acr`, `default`, `dockerhub`, `gcr`, `github`, `gitlab`, `harbor`, `quay`.

Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or `auto` mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default `$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD`)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default `$WERF_FINAL_REPO_DOCKER_HUB_TOKEN`)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default `$WERF_FINAL_REPO_DOCKER_HUB_USERNAME`)

- `--final-repo-github-token`

final-repo GitHub token (default `$WERF_FINAL_REPO_GITHUB_TOKEN`)

- `--final-repo-harbor-password`

final-repo Harbor password (default `$WERF_FINAL_REPO_HARBOR_PASSWORD`)

- `--final-repo-harbor-username`

final-repo Harbor username (default `$WERF_FINAL_REPO_HARBOR_USERNAME`)

- `--final-repo-quay-token`

final-repo quay.io token (default `$WERF_FINAL_REPO_QUAY_TOKEN`)

- `--follow`

Enable follow mode (default `$WERF_FOLLOW`).

The mode allows restarting the command on a new commit.

In development mode (`--dev`), werf restarts the command on any changes (including untracked

files) in the git repository worktree

- `--force-adoption`

Always adopt resources, even if they belong to a different Helm release (default \$WERF_FORCE_ADOPTION or false)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--hooks-status-progress-period`

No-op

- `--ignore-bundle-js`

Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default \$WERF_IGNORE_BUNDLE_JS)

- `--ignore-secret-key`

Disable secrets decryption (default \$WERF_IGNORE_SECRET_KEY)

- `--insecure-helm-dependencies`

Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage

- `--introspect-error`

Introspect failed stage in the state, right after running failed assembly instruction

- `--introspect-stage`

Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`

Kubernetes API server address (default \$WERF_KUBE_API_SERVER)

- `--kube-auth-password`

Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)

- `--kube-auth-provider`

Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)

- `--kube-auth-provider-config`

Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)

- `--kube-auth-username`

Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

- Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)
- `--kube-ca-path`
Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`
Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
- `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`

Print current project directory path (default \$WERF_LOG_PROJECT_DIR)

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--namespace`

Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)

- `--network-parallelism`

Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)

- `--no-final-tracking`

By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default \$WERF_NO_FINAL_TRACKING)

- `--no-install-crds`

Do not install CRDs from "crds/" directories of installed charts (default \$WERF_NO_INSTALL_CRDS)

- `--no-notes`

Don't show release notes at the end of the release (default \$WERF_NO_NOTES)

- `--no-pod-logs`

Disable Pod logs collection and printing (default \$WERF_NO_POD_LOGS or false)

- `--no-remove-manual-changes`

Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)

- `--parallel, -p`

Run in parallel (default \$WERF_PARALLEL or true)

- `--parallel-tasks-limit`

Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)

- `--plan-lifetime`

How long plan artifact is valid

- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)

- `--project-name, -N`

Set a specific project name (default \$WERF_PROJECT_NAME)

- `--provenance-keyring`

Path to keyring containing public keys to verify chart provenance (default

- `$WERF_PROVENANCE_KEYRING)`
- `--provenance-strategy`
Strategy for provenance verifying (default `$WERF_PROVENANCE_STRATEGY`).
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--release`
Use specified Helm release name (default `[[ project ]]`-`[[ env ]]` template or `deploy.helmRelease` custom template from `werf.yaml` or `$WERF_RELEASE`)
- `--release-info-annotations`
Add annotations to release metadata (default `$WERF_RELEASE_INFO_ANNOTATIONS`)
- `--release-label`
Add Helm release labels (can specify multiple). Kind of labels depends on release storage driver.
Format: `labelName=labelValue`.
Also, can be specified with `$WERF_RELEASE_LABEL_*` (e.g.
`$WERF_RELEASE_LABEL_1=labelName1=labelValue1`,
`$WERF_RELEASE_LABEL_2=labelName2=labelValue2`)
- `--release-storage`
How releases should be stored (default `$WERF_RELEASE_STORAGE`)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default `$WERF_RELEASE_STORAGE_SQL_CONNECTION`)
- `--releases-history-max`
Max releases to keep in release storage (`$WERF_RELEASES_HISTORY_MAX` or 5 by default)
- `--render-subchart-notes`
If set, render subchart notes along with the parent (by default `$WERF_RENDER_SUBCHART_NOTES` or false)
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: `ecr`, `acr`, `default`, `dockerhub`, `gcr`, `github`, `gitlab`, `harbor`, `quay`.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)
- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_REQUIRE_BUILT_IMAGES`)
- `--rollback-graph-path`
Save rollback graph path to the specified file (by default `$WERF_ROLLBACK_GRAPH_PATH`).

- Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default \$WERF_RUNTIME_ANNOTATIONS)
 - `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default \$WERF_RUNTIME_LABELS)
 - `--save-build-report`
Save build report (by default \$WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
 - `--save-deploy-report`
Save deploy report (by default \$WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
 - `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
 - `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
 - `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
 - `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
 - `--set-docker-config-json-value`
Shortcut to set current docker config into the .Values.dockerconfigjson
 - `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1, \$WERF_SET_FILE_2=key2=val2)
 - `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_JSON_* (e.g. \$WERF_SET_JSON_1=key1=val1, \$WERF_SET_JSON_2=key2=val2)
 - `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_LITERAL_* (e.g. \$WERF_SET_LITERAL_1=key1=val1, \$WERF_SET_LITERAL_2=key2=val2)
 - `--set-root-json`
Set new keys in arbitrary things in the global root context (""), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with \$WERF_SET_ROOT_JSON_* (e.g. \$WERF_SET_ROOT_JSON_1=key1=val1, \$WERF_SET_ROOT_JSON_2=key2=val2)
 - `--set-runtime-json`

Set new keys in \$.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.

Can specify multiple or separate values with commas: key1=val1,key2=val2.

Also, can be defined with \$WERF_SET_RUNTIME_JSON_* (e.g.

\$WERF_SET_RUNTIME_JSON_1=key1=val1, \$WERF_SET_RUNTIME_JSON_2=key2=val2)

- `--set-string`

Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).

Also, can be defined with \$WERF_SET_STRING_* (e.g. \$WERF_SET_STRING_1=key1=val1,

\$WERF_SET_STRING_2=key2=val2)

- `--sign-cert`

The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)

- `--sign-elf-files`

Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).

When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option

- `--sign-intermediates`

The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)

- `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)

- `--sign-manifest`

Enable image manifest signing (default \$WERF_SIGN_MANIFEST).

When enabled,

the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option

- `--skip-dependencies-repo-refresh, -L`

Do not refresh helm chart repositories locally cached index

- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)

- `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)

- `--skip-tls-verify-kube`

Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or

- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--templates-allow-dns`

Allow performing DNS requests in templating (default `$WERF_TEMPLATES_ALLOW_DNS`)

- `--timeout, -t`

Resources tracking timeout in seconds (`$WERF_TIMEOUT` by default)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)

- `--use-build-report`

Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).

The alias may contain the following shortcuts:

- `%image%`, `%image_slug%` or `%image_safe_slug%` to use the image name (necessary if there is more than one image in the werf config);

- `%image_content_based_tag%` to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also, can be defined with `$WERF_USE_CUSTOM_TAG` (e.g. `$WERF_USE_CUSTOM_TAG="%image%-tag"`)

- `--use-plan`

Use the gzip-compressed JSON plan file from the specified path during release install

- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with

`$WERF_VALUES_*` (e.g. `$WERF_VALUES_1=.helm/values_1.yaml`,
`$WERF_VALUES_2=.helm/values_2.yaml`)

- `--verbose`

Enable verbose output (default `$WERF_VERBOSE`).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (`$WERF_VIRTUAL_MERGE` by default)

- `--without-images`

Disable building of images defined in the `werf.yaml` (if any) and usage of such images in the `.helm/templates` (`$WERF_WITHOUT_IMAGES` or false by default — e.g. enable all images defined in the `werf.yaml` by default)

d8 delivery-kit cr

Usage:

```
d8 delivery-kit cr [options]
```

Authenticate with container registry.

d8 delivery-kit cr login

Usage:

```
d8 delivery-kit cr login registry [options]
```

Login into remote registry.

Options

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--password, -p`
Use specified password for login (default \$WERF_PASSWORD)
- `--password-stdin`
Read password from stdin for login (default \$WERF_PASSWORD_STDIN)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--username, -u`
Use specified username for login (default \$WERF_USERNAME)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit cr logout

Usage:

```
d8 delivery-kit cr logout registry [options]
```

Logout from a remote registry.

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit dismiss

Usage:

```
d8 delivery-kit dismiss [options]
```

Delete werf release from Kubernetes

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)

- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)
- `--deploy-report-path`
Change deploy report path and format (by default `$WERF_DEPLOY_REPORT_PATH` or `".werf-deploy-report.json"` if not set). Extension must be `.json` for JSON format. If extension not specified, then `.json` is used
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like `werf-build`, `werf-converge` and other (default disabled or `WERF_DISABLE_AUTO_HOST_CLEANUP`)
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
- `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage

volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)

- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--hooks-status-progress-period`

No-op

- `--insecure-helm-dependencies`

No-op

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--kube-api-server`

Kubernetes API server address (default \$WERF_KUBE_API_SERVER)

- `--kube-auth-password`

Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)

- `--kube-auth-provider`

Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)

- `--kube-auth-provider-config`

Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)

- `--kube-auth-username`

Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default `$WERF_KUBE_CA_DATA`)
- `--kube-ca-path`
Kubernetes API server CA path (default `$WERF_KUBE_CA_PATH`)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT`)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default `$WERF_KUBE_CERT_DATA`)
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_CLUSTER`)
- `--kube-context-user`
Use user from Kubeconfig for current context (default `$WERF_KUBE_CONTEXT_USER`)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with `$WERF_KUBE_IMPERSONATE_GROUP_*` environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_UID`)
- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default `$WERF_KUBE_IMPERSONATE_USER`)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY`)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default `$WERF_KUBE_KEY_DATA`)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default `$WERF_KUBE_PROXY_URL`)
- `--kube-qps-limit`
Kubernetes client QPS limit (default `$WERF_KUBE_QPS_LIMIT` or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default `$WERF_KUBE_REQUEST_TIMEOUT`)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default `$WERF_KUBE_TLS_SERVER`)
- `--kube-token`
Kubernetes bearer token used for authentication (default `$WERF_KUBE_TOKEN`)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default `$WERF_KUBE_TOKEN_PATH`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.

- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)
- `--network-parallelism`
Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`
By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default \$WERF_NO_FINAL_TRACKING)
- `--no-pod-logs`
Disable Pod logs collection and printing (default \$WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or \$WERF_RELEASE)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`
Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`

Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)

- `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--timeout, -t`
Resources tracking timeout in seconds (\$WERF_TIMEOUT by default)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--use-deploy-report`
Use deploy report, previously saved with --save-deploy-report (by default \$WERF_USE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).
- `--with-hooks`
Delete Helm Release hooks getting from existing revisions (default \$WERF_WITH_HOOKS or true)
- `--with-namespace`

Delete Kubernetes Namespace after purging Helm Release (default \$WERF_WITH_NAMESPACE)

d8 delivery-kit export

Usage:

```
d8 delivery-kit export [IMAGE_NAME...] [options]
```

Export images

Options

- `--add-label`
Add label to exported images (can specify multiple).
Format: `labelName=labelValue[labelName=labelValue ...]`. The default separator is a newline ("`\n`"), but it can be customized using the `--add-label-separator` flag.
Also, can be specified with `$WERF_EXPORT_ADD_LABEL_*` (e.g. `$WERF_EXPORT_ADD_LABEL_1=labelName1=labelValue1`, `$WERF_EXPORT_ADD_LABEL_2=labelName2=labelValue2`)
- `--add-label-separator`
Separator for `--add-label` values (default `$WERF_EXPORT_ADD_LABEL_SEPARATOR` or "`\n`")
- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)

- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read and pull images from the specified repo
- `--elf-pgp-private-key-base64`
Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`
PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--insecure-helm-dependencies`

No-op

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage

- `--introspect-error`

Introspect failed stage in the state, right after running failed assembly instruction

- `--introspect-stage`

Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact
IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name
STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`

Print current project directory path (default \$WERF_LOG_PROJECT_DIR)

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

- Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)
- `--sign-elf-files`
Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).
When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)
- `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)

- `--sign-manifest`

Enable image manifest signing (default \$WERF_SIGN_MANIFEST).

When enabled,

the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option

- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if `--repo` is not specified, or
- https://delivery-sync.deckhouse.ru if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tag`

Set a tag template (can specify multiple).

It is necessary to use image name shortcut %image% or %image_slug% if multiple images are exported (e.g. REPO:TAG-%image% or REPO-%image%:TAG)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

Use build report, previously saved with `--save-build-report` (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with `--build-report-path`

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

d8 delivery-kit helm

Usage:

```
d8 delivery-kit helm [options]
```

Manage application deployment with helm

Options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or

- ~/.docker (in the order of priority)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`
No-op
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--releases-history-max`
Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm create

Usage:

```
d8 delivery-kit helm create NAME [options]
```

create a new chart with the given name

Options

- `--starter`, `-p`
the name or absolute path to Helm starter scaffold

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace`, `-n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm dependency

Aliases: `dep`, `dependencies`**Usage:**`d8 delivery-kit helm dependency update|build|list [options]`

manage a chart's dependencies

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm dependency build**Usage:**

```
d8 delivery-kit helm dependency update|build|list build CHART [options]
```

rebuild the charts/ directory based on the Chart.lock file

Options

- `--keyring`
keyring containing public keys
- `--skip-refresh`
do not refresh the local repository cache
- `--verify`
verify the packages against signatures

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit helm dependency list

Alias: `ls`

Usage:

```
d8 delivery-kit helm dependency update|build|list list CHART [options]
```

list the dependencies for the given chart

Options

- `--max-col-width`
maximum column width for output table

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm dependency update

Alias: `up`

Usage:

```
d8 delivery-kit helm dependency update|build|list update CHART [options]
```

update charts/ based on the contents of Chart.yaml

Options

- `--keyring`
keyring containing public keys
- `--skip-refresh`
do not refresh the local repository cache
- `--verify`
verify the packages against signatures

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm env

Usage:

```
d8 delivery-kit helm env [options]
```

helm client environment information

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get

Usage:

```
d8 delivery-kit helm get [options]
```

download extended information of a named release

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get all

Usage:

```
d8 delivery-kit helm get all RELEASE_NAME [options]
```

download all information for a named release

Options

- `--revision`
get the named release with revision
- `--template`
go template for formatting the output, eg: `{{.Release.Name}}`

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request

- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get hooks

Usage:

```
d8 delivery-kit helm get hooks RELEASE_NAME [options]
```

download all hooks for a named release

Options

- `--revision`
get the named release with revision

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`

- namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get manifest

Usage:

```
d8 delivery-kit helm get manifest RELEASE_NAME [options]
```

download the manifest for a named release

Options

- `--revision`
get the named release with revision

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get metadata

Usage:

```
d8 delivery-kit helm get metadata RELEASE_NAME [options]
```

This command fetches metadata for a given release

Options

- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`
specify release revision

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`

- Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get notes

Usage:

```
d8 delivery-kit helm get notes RELEASE_NAME [options]
```

download the notes for a named release

Options

- `--revision`
get the named release with revision

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get values

Usage:

```
d8 delivery-kit helm get values RELEASE_NAME [options]
```

download the values file for a named release

Options

- `--all, -a`
dump all (computed) values
- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`
get the named release with revision

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm get-autogenerated-values

Usage:

```
d8 delivery-kit helm get-autogenerated-values [IMAGE_NAME...] [options]
```

Get service values yaml generated by werf for helm chart during deploy

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--backend-network`

Network mode for the build containers (\$WERF_BACKEND_NETWORK or nothing by default)

- `--build-report-path`

Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

- Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

 - `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`

Use specified environment (default \$WERF_ENV)
- `--final-images-only`

Process final images only (\$WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--introspect-before-error`
Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`
Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`
Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

- * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
- * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)

- `--quiet`

Disable explanatory output (default `$WERF_QUIET`).

- `--repo`

Container registry storage address (default `$WERF_REPO`)

- `--repo-container-registry`

Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)

- `--repo-docker-hub-token`

repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)

- `--repo-docker-hub-username`

repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)

- `--repo-github-token`

repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)

- `--repo-harbor-password`

repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)

- `--repo-harbor-username`

repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)

- `--repo-quay-token`

repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)

- `--require-built-images, -Z`

Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_REQUIRE_BUILT_IMAGES`)

- `--save-build-report`

Save build report (by default `$WERF_SAVE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)

- `--set-docker-config-json-value`

Shortcut to set current docker config into the `.Values.dockerconfigjson`

- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`, `$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).

Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`

- `--stub-tags`

Use stubs instead of real tags (default `$WERF_STUB_TAGS`)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

Use build report, previously saved with --save-build-report (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).

The alias may contain the following shortcuts:

- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also, can be defined with \$WERF_USE_CUSTOM_TAG (e.g. \$WERF_USE_CUSTOM_TAG="%image%-tag")

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

Global options

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-namespace

Usage:

```
d8 delivery-kit helm get-namespace [options]
```

Print Kubernetes Namespace that will be used in current configuration with specified params.

Options

- `--allow-includes-update`

- Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140

- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--namespace, -n`
namespace scope for this request
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-release

Usage:

```
d8 delivery-kit helm get-release [options]
```

Print Helm Release name that will be used in current configuration with specified params.

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--namespace`

Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm history**Alias:** `hist`**Usage:**

```
d8 delivery-kit helm history RELEASE_NAME [options]
```

fetch release history

Options

- `--max`
maximum number of revision to include in history
- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`

- Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm install

Usage:

```
d8 delivery-kit helm install [NAME] [CHART] [options]
```

install a chart

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g. \$WERF_ADD_ANNOTATION_1=annoName1=annoValue1, \$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`
Add label to deploying resources (can specify multiple).
Format: labelName=labelValue.
Also, can be specified with \$WERF_ADD_LABEL_* (e.g. \$WERF_ADD_LABEL_1=labelName1=labelValue1, \$WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`
if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used
- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`
allow deletion of new resources created in this installation when install fails
- `--create-namespace`

- create the release namespace if not present
- `--dependency-update`
update dependencies if they are missing before installing the chart
- `--deploy-report-path`
save deploy report in JSON to the specified path
- `--description`
add a custom description
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If `--version` is set, this is ignored
- `--disable-default-secret-values`
Do not use secret values from the default `.helm/secret-values.yaml` file (default `$WERF_DISABLE_DEFAULT_SECRET_VALUES` or `false`)
- `--disable-openapi-validation`
if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`
simulate an install. If `--dry-run` is set with no option being specified or as `'--dry-run=client'`, it will not attempt cluster connections. Setting `'--dry-run=server'` allows attempting cluster connections.
- `--enable-dns`
enable DNS lookups when rendering templates
- `--force`
force resource updates through a replacement strategy
- `--generate-name, -g`
generate the name (and omit the `NAME` parameter)
- `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--labels, -l`
Labels that would be added to release metadata. Should be divided by comma.
- `--name-template`
specify template used to name the release
- `--no-hooks`
prevent hooks from running during install
- `--output, -o`
prints the output in the specified format. Allowed values: `table`, `json`, `yaml`
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--post-renderer`
the path to an executable to be used for post rendering. If it exists in `$PATH`, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`
an argument to the `post-renderer` (can specify multiple)
- `--render-subchart-notes`
if set, render subchart notes along with the parent

- `--replace`
re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`
chart repository url where to locate the requested chart
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`
set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`
set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`
set a literal STRING value on the command line
- `--set-string`
set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`
if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--timeout`
time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`
chart repository username where to locate the requested chart
- `--values, -f`
specify values in a YAML file or a URL (can specify multiple)
- `--verify`
verify the package before using it
- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`
if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`
if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm lint

Usage:

```
d8 delivery-kit helm lint PATH [options]
```

examine a chart for possible issues

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: annoName=annoValue.

Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.

\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: `labelName=labelValue`.

Also, can be specified with `$WERF_ADD_LABEL_*` (e.g.

`$WERF_ADD_LABEL_1=labelName1=labelValue1`,

`$WERF_ADD_LABEL_2=labelName2=labelValue2`)

- `--disable-default-secret-values`

Do not use secret values from the default `.helm/secret-values.yaml` file (default

`$WERF_DISABLE_DEFAULT_SECRET_VALUES` or `false`)

- `--env`

Use specified environment (default `$WERF_ENV`)

- `--ignore-secret-key`

Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)

- `--kube-version`

Kubernetes version used for capabilities and deprecation checks

- `--quiet`

print only warnings and errors

- `--secret-key`

Secret key (default `$WERF_SECRET_KEY`)

- `--secret-values`

Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with

`$WERF_SECRET_VALUES_*` (e.g. `$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml`,

`$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml`)

- `--set`

set values on the command line (can specify multiple or separate values with commas:

`key1=val1,key2=val2`)

- `--set-file`

set values from respective files specified via the command line (can specify multiple or separate values with commas: `key1=path1,key2=path2`)

- `--set-json`

set JSON values on the command line (can specify multiple or separate values with commas:

`key1=jsonval1,key2=jsonval2`)

- `--set-literal`

set a literal STRING value on the command line

- `--set-string`

set STRING values on the command line (can specify multiple or separate values with commas:

`key1=val1,key2=val2`)

- `--strict`

fail on lint warnings

- `--values, -f`

specify values in a YAML file or a URL (can specify multiple)

- `--with-subcharts`

lint dependent charts

Global options

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)

- `--kube-config-base64`

Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm list

Alias: `ls`

Usage:

```
d8 delivery-kit helm list [options]
```

list releases

Options

- `--all, -a`

show all releases without any filter applied

- `--all-namespaces, -A`

list releases across all namespaces

- `--date, -d`

sort by release date

- `--deployed`

show deployed releases. If no other is specified, this will be automatically enabled

- `--failed`

show failed releases

- `--filter, -f`

a regular expression (Perl compatible). Any releases that match the expression will be included in the results

- `--max, -m`

maximum number of releases to fetch

- `--no-headers`
don't print headers when using the default output format
- `--offset`
next release index in the list, used to offset from start value
- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml
- `--pending`
show pending releases
- `--reverse, -r`
reverse the sort order
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Works only for secret(default) and configmap storage backends.
- `--short, -q`
output short (quiet) listing format
- `--superseded`
show superseded releases
- `--time-format`
format time using golang time formatter. Example: `--time-format "2006-01-02 15:04:05Z0700"`
- `--uninstalled`
show uninstalled releases (if 'helm uninstall --keep-history' was used)
- `--uninstalling`
show releases that are currently being uninstalled

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`

- Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm package

Usage:

```
d8 delivery-kit helm package [CHART_PATH] [...] [options]
```

package a chart directory into a chart archive

Options

- `--app-version`
set the appVersion on the chart to this version
- `--dependency-update, -u`
update dependencies from "Chart.yaml" to dir "charts/" before packaging
- `--destination, -d`
location to write the chart.
- `--key`
name of the key to use when signing. Used if --sign is true
- `--keyring`
location of a public keyring
- `--passphrase-file`
location of a file which contains the passphrase for the signing key. Use "-" in order to read from stdin.
- `--sign`
use a PGP private key to sign this package
- `--version`
set the version on the chart to this semver version

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm plugin

Usage:

```
d8 delivery-kit helm plugin [options]
```

install, list, or uninstall Helm plugins

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

- `--log-debug`
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-pretty`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-quiet`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-terminal-width`
Disable explanatory output (default \$WERF_LOG_QUIET).
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm plugin install

Alias: `add`

Usage:

```
d8 delivery-kit helm plugin install [options] <path|url>...
```

install one or more Helm plugins

Options

- `--version`
specify a version constraint. If this is not specified, the latest version is installed

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm plugin list

Alias: `ls`

Usage:

```
d8 delivery-kit helm plugin list [options]
```

list installed Helm plugins

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

- Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm plugin uninstall

Aliases: `rm`, `remove`

Usage:

```
d8 delivery-kit helm plugin uninstall <plugin>... [options]
```

uninstall one or more Helm plugins

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

- Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm plugin update

Alias: `up`

Usage:

```
d8 delivery-kit helm plugin update <plugin>... [options]
```

update one or more Helm plugins

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`

- Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm pull

Alias: `fetch`

Usage:

```
d8 delivery-kit helm pull [chart URL | repo/chartname] [...] [options]
```

download a chart from a repository and (optionally) unpack it in local directory

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--destination, -d`
location to write the chart. If this and untardir are specified, untardir is appended to this
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored.
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--prov`

- `--repo`
chart repository url where to locate the requested chart
- `--untar`
if set to true, will untar the chart after downloading it
- `--untardir`
if untar is specified, this flag specifies the name of the directory into which the chart is expanded
- `--username`
chart repository username where to locate the requested chart
- `--verify`
verify the package before using it
- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm registry

Usage:

```
d8 delivery-kit helm registry [options]
```

login to or logout from a registry

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

`$WERF_STATUS_PROGRESS_PERIOD_SECONDS` or 5 seconds

- `--verbose`

Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit helm registry login

Usage:

```
d8 delivery-kit helm registry login [host] [options]
```

login to a registry

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify registry client using this SSL certificate file
- `--insecure`
allow connections to TLS registry without certs
- `--key-file`
identify registry client using this SSL key file
- `--password`, `-p`
registry password or identity token
- `--password-stdin`
read password or identity token from stdin
- `--username`, `-u`
registry username

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`

- * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to `$WERF_STATUS_PROGRESS_PERIOD_SECONDS` or 5 seconds
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit helm registry logout

Usage:

```
d8 delivery-kit helm registry logout [host] [options]
```

logout from a registry

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).

- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to `$WERF_STATUS_PROGRESS_PERIOD_SECONDS` or 5 seconds
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit helm repo

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] [options]
```

add, list, remove, update, and index chart repositories

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm repo add

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] add [NAME] [URL] [options]
```

add a chart repository

Options

- `--allow-deprecated-repos`

by default, this command will not allow adding official repos that have been permanently deleted.

This disables that behavior

- `--ca-file`

verify certificates of HTTPS-enabled servers using this CA bundle

- `--cert-file`

identify HTTPS client using this SSL certificate file

- `--force-update`

replace (overwrite) the repo if it already exists

- `--insecure-skip-tls-verify`

skip tls certificate checks for the repository

- `--key-file`

identify HTTPS client using this SSL key file

- `--no-update`

Ignored. Formerly, it would disabled forced updates. It is deprecated by force-update.

- `--pass-credentials`

pass credentials to all domains

- `--password`

chart repository password

- `--password-stdin`

read chart repository password from stdin

- `--username`

chart repository username

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm repo index

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] index [DIR] [options]
```

generate an index file given a directory containing packaged charts

Options

- `--json`
output in JSON format
- `--merge`
merge the generated index into the given index
- `--url`
url of chart repository

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm repo list

Alias: `ls`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] list [options]
```

list chart repositories

Options

- `--output, -o`

prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

- `No-op`
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm repo remove

Alias: `rm`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] remove [REP01 [REP02 ...]] [options]
```

remove one or more chart repositories

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
- `No-op`
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm repo update

Alias: `up`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] update [REP01 [REP02 ...]] [options]
```

update information of available charts locally from chart repositories

Options

- `--fail-on-repo-update-fail`

update fails if any of the repository updates fail

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

- No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm rollback

Usage:

```
d8 delivery-kit helm rollback <RELEASE> [REVISION] [options]
```

roll back a release to a previous revision

Options

- `--cleanup-on-fail`
allow deletion of new resources created in this rollback when rollback fails
- `--deploy-report-path`
save deploy report in JSON to the specified path
- `--dry-run`
simulate a rollback

- `--force`
force resource update through delete/recreate if needed
- `--history-max`
limit the maximum number of revisions saved per release. Use 0 for no limit
- `--no-hooks`
prevent hooks from running during rollback
- `--recreate-pods`
performs pods restart for the resource if applicable
- `--timeout`
time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--wait`
if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as `--timeout`
- `--wait-for-jobs`
if set and `--wait` enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as `--timeout`

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--namespace, -n`

- namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm search

Usage:

```
d8 delivery-kit helm search [keyword] [options]
```

search for a keyword in charts

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm search hub

Usage:

```
d8 delivery-kit helm search [keyword] hub [KEYWORD] [options]
```

search for charts in the Artifact Hub or your own hub instance

Options

- `--endpoint`
Hub instance to query for charts
- `--fail-on-no-result`
search fails if no results are found
- `--list-repo-url`
print charts repository URL
- `--max-col-width`
maximum column width for output table
- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`

- Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm search repo

Usage:

```
d8 delivery-kit helm search [keyword] repo [keyword] [options]
```

search repositories for a keyword in charts

Options

- `--devel`
use development versions (alpha, beta, and release candidate releases), too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--fail-on-no-result`
search fails if no results are found
- `--max-col-width`
maximum column width for output table
- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml
- `--regexp, -r`
use regular expressions for searching repositories you have added
- `--version`
search using semantic versioning constraints on repositories you have added
- `--versions, -l`
show the long listing, with each version of each chart on its own line, for repositories you have added

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm secret

Usage:

```
d8 delivery-kit helm secret [options]
```

Work with secrets

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm secret decrypt

Usage:

```
d8 delivery-kit helm secret decrypt [options]
```

Decrypt data

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--output-file-path, -o`

Write to file instead of stdout

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret encrypt

Usage:

```
d8 delivery-kit helm secret encrypt [options]
```

Encrypt data

Options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to file instead of stdout
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file

Usage:

```
d8 delivery-kit helm secret file [options]
```

Work with secret files

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm secret file decrypt

Usage:

```
d8 delivery-kit helm secret file decrypt [FILE_PATH] [options]
```

Decrypt secret file data

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to file instead of stdout
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--namespace, -n`
namespace scope for this request
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file edit

Usage:

```
d8 delivery-kit helm secret file edit FILE_PATH [options]
```

Edit or create new secret file

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file encrypt

Usage:

```
d8 delivery-kit helm secret file encrypt [FILE_PATH] [options]
```

Encrypt file data

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to file instead of stdout
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op

- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--namespace, -n`
namespace scope for this request
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret generate-secret-key

Usage:

```
d8 delivery-kit helm secret generate-secret-key [options]
```

Generate hex encryption key

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret rotate-secret-key

Usage:

```
d8 delivery-kit helm secret rotate-secret-key [EXTRA_SECRET_VALUES_FILE_PATH...] [options]
```

Regenerate secret files with new secret key

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default

\$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values

Usage:

```
d8 delivery-kit helm secret values [options]
```

Work with secret values files

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm secret values decrypt

Usage:

```
d8 delivery-kit helm secret values decrypt [FILE_PATH] [options]
```

Decrypt secret values file data

Options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to file instead of stdout
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values edit

Usage:

```
d8 delivery-kit helm secret values edit FILE_PATH [options]
```

Edit or create new secret values file

Options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--namespace, -n`

namespace scope for this request

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values encrypt

Usage:

```
d8 delivery-kit helm secret values encrypt [FILE_PATH] [options]
```

Encrypt values file data

Options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--output-file-path, -o`
Write to file instead of stdout
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--namespace, -n`
namespace scope for this request
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm show

Alias: `inspect`

Usage:

```
d8 delivery-kit helm show [options]
```

show information of a chart

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm show all**Usage:**

```
d8 delivery-kit helm show all [CHART] [options]
```

show all information of the chart

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If `--version` is set, this is ignored
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--repo`
chart repository url where to locate the requested chart
- `--username`
chart repository username where to locate the requested chart
- `--verify`
verify the package before using it
- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm show chart

Usage:

```
d8 delivery-kit helm show chart [CHART] [options]
```

show the chart's definition

Options

- `--ca-file`

verify certificates of HTTPS-enabled servers using this CA bundle

- `--cert-file`

identify HTTPS client using this SSL certificate file

- `--devel`

use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored

- `--insecure-skip-tls-verify`

skip tls certificate checks for the chart download

- `--key-file`

identify HTTPS client using this SSL key file

- `--keyring`

location of public keys used for verification

- `--pass-credentials`

pass credentials to all domains

- `--password`

chart repository password where to locate the requested chart

- `--plain-http`

use insecure HTTP connections for the chart download

- `--repo`

chart repository url where to locate the requested chart

- `--username`

chart repository username where to locate the requested chart

- `--verify`

verify the package before using it

- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm show crds

Usage:

```
d8 delivery-kit helm show crds [CHART] [options]
```

show the chart's CRDs

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If `--version` is set, this is ignored
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--repo`
chart repository url where to locate the requested chart
- `--username`
chart repository username where to locate the requested chart
- `--verify`
verify the package before using it
- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-quiet`

- Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm show readme

Usage:

```
d8 delivery-kit helm show readme [CHART] [options]
```

show the chart's README

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--repo`
chart repository url where to locate the requested chart
- `--username`
chart repository username where to locate the requested chart
- `--verify`
verify the package before using it

- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm show values

Usage:

```
d8 delivery-kit helm show values [CHART] [options]
```


show the chart's values

Options

- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
identify HTTPS client using this SSL certificate file
- `--devel`
use development versions, too. Equivalent to version '>0.0.0-0'. If `--version` is set, this is ignored
- `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
- `--jsonpath`
supply a JSONPath expression to filter the output
- `--key-file`
identify HTTPS client using this SSL key file
- `--keyring`
location of public keys used for verification
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--repo`
chart repository url where to locate the requested chart
- `--username`
chart repository username where to locate the requested chart
- `--verify`
verify the package before using it
- `--version`
specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`

- Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm status

Usage:

```
d8 delivery-kit helm status RELEASE_NAME [options]
```

display the status of the named release

Options

- `--output, -o`
prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`
if set, display the status of the named release with revision
- `--show-desc`
if set, display the description message of the named release
- `--show-resources`
if set, display the resources of the named release

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm template

Usage:

```
d8 delivery-kit helm template [NAME] [CHART] [options]
```

locally render templates

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g. \$WERF_ADD_ANNOTATION_1=annoName1=annoValue1, \$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`
Add label to deploying resources (can specify multiple).
Format: labelName=labelValue.
Also, can be specified with \$WERF_ADD_LABEL_* (e.g. \$WERF_ADD_LABEL_1=labelName1=labelValue1, \$WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--api-versions, -a`
Kubernetes api versions used for Capabilities.APIVersions
- `--atomic`

- if set, the installation process deletes the installation on failure. The `--wait` flag will be set automatically if `--atomic` is used
- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
 - `--cert-file`
identify HTTPS client using this SSL certificate file
 - `--cleanup-on-fail`
allow deletion of new resources created in this installation when install fails
 - `--create-namespace`
create the release namespace if not present
 - `--dependency-update`
update dependencies if they are missing before installing the chart
 - `--deploy-report-path`
save deploy report in JSON to the specified path
 - `--description`
add a custom description
 - `--devel`
use development versions, too. Equivalent to version `'>0.0.0-0'`. If `--version` is set, this is ignored
 - `--disable-default-secret-values`
Do not use secret values from the default `.helm/secret-values.yaml` file (default `$WERF_DISABLE_DEFAULT_SECRET_VALUES` or `false`)
 - `--disable-openapi-validation`
if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
 - `--dry-run`
simulate an install. If `--dry-run` is set with no option being specified or as `'--dry-run=client'`, it will not attempt cluster connections. Setting `'--dry-run=server'` allows attempting cluster connections.
 - `--enable-dns`
enable DNS lookups when rendering templates
 - `--force`
force resource updates through a replacement strategy
 - `--generate-name, -g`
generate the name (and omit the `NAME` parameter)
 - `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
 - `--include-crds`
include CRDs in the templated output
 - `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
 - `--is-upgrade`
set `.Release.IsUpgrade` instead of `.Release.IsInstall`
 - `--key-file`
identify HTTPS client using this SSL key file
 - `--keyring`
location of public keys used for verification
 - `--kube-version`
Kubernetes version used for `Capabilities.KubeVersion`
 - `--labels, -l`
Labels that would be added to release metadata. Should be divided by comma.
 - `--name-template`
specify template used to name the release
 - `--no-hooks`
prevent hooks from running during install

- `--output-dir`
writes the executed templates to files in output-dir instead of stdout
- `--pass-credentials`
pass credentials to all domains
- `--password`
chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--post-renderer`
the path to an executable to be used for post rendering. If it exists in \$PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`
an argument to the post-renderer (can specify multiple)
- `--release-name`
use release name in the output-dir path.
- `--render-subchart-notes`
if set, render subchart notes along with the parent
- `--replace`
re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`
chart repository url where to locate the requested chart
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`
set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`
set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`
set a literal STRING value on the command line
- `--set-string`
set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--show-only, -s`
only show manifests rendered from the given templates
- `--skip-crds`
if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--skip-tests`
skip tests from templated output
- `--timeout`
time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`
chart repository username where to locate the requested chart
- `--validate`

validate your manifests against the Kubernetes cluster you are currently pointing at. This is the same validation performed on an install

- `--values, -f`

specify values in a YAML file or a URL (can specify multiple)

- `--verify`

verify the package before using it

- `--version`

specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

- `--wait`

if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout

- `--wait-for-jobs`

if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

- namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm test

Usage:

```
d8 delivery-kit helm test [RELEASE] [options]
```

run tests for a release

Options

- `--filter`
specify tests by attribute (currently "name") using attribute=value syntax or '!attribute=value' to exclude a test (can specify multiple or separate values with commas: name=test1,name=test2)
- `--logs`
dump the logs from test pods (this runs after all tests are complete, but before any cleanup)
- `--timeout`
time to wait for any individual Kubernetes operation (like Jobs for hooks)

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm uninstall

Aliases: `del`, `delete`, `un`

Usage:

```
d8 delivery-kit helm uninstall RELEASE_NAME [...] [options]
```

uninstall a release

Options

- `--cascade`

Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents. Defaults to background.

- `--description`

add a custom description

- `--dry-run`

simulate a uninstall

- `--ignore-not-found`

Treat "release not found" as a successful uninstall

- `--keep-history`

remove all associated resources and mark the release as deleted, but retain the release history

- `--no-hooks`

prevent hooks from running during uninstallation

- `--timeout`

time to wait for any individual Kubernetes operation (like Jobs for hooks)

- `--wait`

if set, will wait until all the resources are deleted before returning. It will wait for as long as --timeout

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm upgrade

Usage:

```
d8 delivery-kit helm upgrade [RELEASE] [CHART] [options]
```

upgrade a release

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g. \$WERF_ADD_ANNOTATION_1=annoName1=annoValue1, \$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`
Add label to deploying resources (can specify multiple).
Format: labelName=labelValue.
Also, can be specified with \$WERF_ADD_LABEL_* (e.g. \$WERF_ADD_LABEL_1=labelName1=labelValue1, \$WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

- if set, upgrade process rolls back changes made in case of failed upgrade. The `--wait` flag will be set automatically if `--atomic` is used
- `--ca-file`
verify certificates of HTTPS-enabled servers using this CA bundle
 - `--cert-file`
identify HTTPS client using this SSL certificate file
 - `--cleanup-on-fail`
allow deletion of new resources created in this upgrade when upgrade fails
 - `--create-namespace`
if `--install` is set, create the release namespace if not present
 - `--dependency-update`
update dependencies if they are missing before installing the chart
 - `--deploy-report-path`
save deploy report in JSON to the specified path
 - `--description`
add a custom description
 - `--devel`
use development versions, too. Equivalent to version `'>0.0.0-0'`. If `--version` is set, this is ignored
 - `--disable-default-secret-values`
Do not use secret values from the default `.helm/secret-values.yaml` file (default `$WERF_DISABLE_DEFAULT_SECRET_VALUES` or `false`)
 - `--disable-openapi-validation`
if set, the upgrade process will not validate rendered templates against the Kubernetes OpenAPI Schema
 - `--dry-run`
simulate an install. If `--dry-run` is set with no option being specified or as `'--dry-run=client'`, it will not attempt cluster connections. Setting `'--dry-run=server'` allows attempting cluster connections.
 - `--enable-dns`
enable DNS lookups when rendering templates
 - `--force`
force resource updates through a replacement strategy
 - `--history-max`
limit the maximum number of revisions saved per release. Use 0 for no limit
 - `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
 - `--insecure-skip-tls-verify`
skip tls certificate checks for the chart download
 - `--install, -i`
if a release by this name doesn't already exist, run an install
 - `--key-file`
identify HTTPS client using this SSL key file
 - `--keyring`
location of public keys used for verification
 - `--labels, -l`
Labels that would be added to release metadata. Should be separated by comma. Original release labels will be merged with upgrade labels. You can unset label using `null`.
 - `--no-hooks`
disable pre/post upgrade hooks
 - `--output, -o`
prints the output in the specified format. Allowed values: `table`, `json`, `yaml`
 - `--pass-credentials`
pass credentials to all domains
 - `--password`

- chart repository password where to locate the requested chart
- `--plain-http`
use insecure HTTP connections for the chart download
- `--post-renderer`
the path to an executable to be used for post rendering. If it exists in \$PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`
an argument to the post-renderer (can specify multiple)
- `--recreate-pods`
performs pods restart for the resource if applicable
- `--render-subchart-notes`
if set, render subchart notes along with the parent
- `--repo`
chart repository url where to locate the requested chart
- `--reset-then-reuse-values`
when upgrading, reset the values to the ones built into the chart, apply the last release's values and merge in any overrides from the command line via --set and -f. If '--reset-values' or '--reuse-values' is specified, this is ignored
- `--reset-values`
when upgrading, reset the values to the ones built into the chart
- `--reuse-values`
when upgrading, reuse the last release's values and merge in any overrides from the command line via --set and -f. If '--reset-values' is specified, this is ignored
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`
set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`
set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`
set a literal STRING value on the command line
- `--set-string`
set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`
if set, no CRDs will be installed when an upgrade is performed with install flag enabled. By default, CRDs are installed if not already present, when an upgrade is performed with install flag enabled
- `--timeout`
time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`
chart repository username where to locate the requested chart
- `--values, -f`
specify values in a YAML file or a URL (can specify multiple)
- `--verify`

verify the package before using it

- `--version`

specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

- `--wait`

if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as `--timeout`

- `--wait-for-jobs`

if set and `--wait` enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as `--timeout`

Global options

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)

- `--kube-config-base64`

Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)

- `--kube-context`

Kubernetes config context (default `$WERF_KUBE_CONTEXT`)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default `$WERF_LOG_COLOR_MODE` or auto mode.

- `--log-debug`

Enable debug (default `$WERF_LOG_DEBUG`).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).

- `--log-quiet`

Disable explanatory output (default `$WERF_LOG_QUIET`).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* `$WERF_LOG_TERMINAL_WIDTH`

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).

- `--log-time-format`

Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).

- `--log-verbose`

Enable verbose output (default `$WERF_LOG_VERBOSE`).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default `$WERF_QUIET`).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm verify

Usage:

```
d8 delivery-kit helm verify PATH [options]
```

verify that a chart at the given path has been signed and is valid

Options

- `--keyring`

keyring containing public keys

Global options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--hooks-status-progress-period`

No-op

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`

namespace scope for this request

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit helm version

Usage:

```
d8 delivery-kit helm version [options]
```

print the client version information

Options

- `--client, -c`
display client version information
- `--short`
print the version number
- `--template`
template for version string format

Global options

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--hooks-status-progress-period`
No-op
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--namespace, -n`
namespace scope for this request
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to \$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit host

Usage:

```
d8 delivery-kit host [options]
```

Work with werf cache and data of all projects on the host machine

d8 delivery-kit host cleanup

Usage:

```
d8 delivery-kit host cleanup [options]
```

Cleanup old unused werf cache and data of all projects on host machine.

Options

- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)

- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`
Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--force`
Force deletion of images which are being used by some containers (default \$WERF_FORCE)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit host purge

Usage:

```
d8 delivery-kit host purge [options]
```

Purge werf images, cache and other data for all projects on host machine

Options

- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--dry-run`
Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--force`
First remove containers that use werf docker images which are going to be deleted
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit includes

Usage:

```
d8 delivery-kit includes [options]
```

Work with werf includes

d8 delivery-kit includes get-file

Usage:

```
d8 delivery-kit includes get-file [FILE_NAME] [options]
```

Display file content that will be used by werf according to the includes overlay rules.

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--follow`
Enable follow mode (default \$WERF_FOLLOW).
The mode allows restarting the command on a new commit.
In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`

- Disable explanatory output (default \$WERF_LOG_QUIET).
 - `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
 - `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
 - `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
 - `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
 - `--loose-giterminism`
Loose werf giterminism mode restrictions
 - `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
 - `--quiet`
Disable explanatory output (default \$WERF_QUIET).
 - `--ssh-key`
Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
 - `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
 - `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit includes ls-files

Usage:

```
d8 delivery-kit includes ls-files [GLOB...] [options]
```

List files in the project according to the includes overlay rules.

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)

- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)

- `--env`

Use specified environment (default `$WERF_ENV`)

- `--filter`

Filter by source, e.g. `--filter=source=local,remoteRepo` (default `$WERF_INCLUDES_LIST_FILTER` or all sources).

- `--follow`

Enable follow mode (default `$WERF_FOLLOW`).

The mode allows restarting the command on a new commit.

In development mode (`--dev`), `werf` restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`

Use specified git work tree dir (default `$WERF_WORK_TREE` or lookup for directory that contains `.git` in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default `$WERF_GITERMINISM_CONFIG` or `werf-giterminism.yaml` in working directory)

- `--home-dir`

Use specified dir to store `werf` cache files and dirs (default `$WERF_HOME` or `~/.werf`)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes. Default `$WERF_LOG_COLOR_MODE` or auto mode.

- `--log-debug`

Enable debug (default `$WERF_LOG_DEBUG`).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).

- `--log-project-dir`

Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)

- `--log-quiet`

Disable explanatory output (default `$WERF_LOG_QUIET`).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* `$WERF_LOG_TERMINAL_WIDTH`

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).

- `--log-time-format`

Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).

- `--log-verbose`

Enable verbose output (default `$WERF_LOG_VERBOSE`).

- `--loose-giterminism`

Loose `werf` giterminism mode restrictions

- `--platform`

Enable platform emulation when building images with `werf`, format: `OS/ARCH[/VARIANT]` (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)

- `--quiet`

Disable explanatory output (default `$WERF_QUIET`).

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`, `$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).

Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)

- `--verbose`

Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit includes update

Usage:

```
d8 delivery-kit includes update [options]
```

Create or update includes lock file (default: `werf-includes.lock`).

Options

- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)

- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)

- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)

- `--env`

Use specified environment (default `$WERF_ENV`)

- `--follow`

Enable follow mode (default `$WERF_FOLLOW`).

The mode allows restarting the command on a new commit.

In development mode (`--dev`), werf restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`

Use specified git work tree dir (default `$WERF_WORK_TREE` or lookup for directory that contains `.git` in the current or parent directories)

- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default `$WERF_GITERMINISM_CONFIG` or `werf-giterminism.yaml` in working directory)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`

Print current project directory path (default \$WERF_LOG_PROJECT_DIR)

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit kube-run

Usage:

```
d8 delivery-kit kube-run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image in Kubernetes

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: annoName=annoValue.

Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.

\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: `labelName=labelValue`.

Also, can be specified with `$WERF_ADD_LABEL_*` (e.g.

`$WERF_ADD_LABEL_1=labelName1=labelValue1`,

`$WERF_ADD_LABEL_2=labelName2=labelValue2`)

- `--allow-includes-update`

Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)

- `--annotate-layers-with-dm-verity-root-hash`

Enable annotation of image layers with dm-verity root hash (default

`$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)

- `--auto-pull-secret`

Automatically create docker config secret in the namespace and plug it via pod's `imagePullSecrets` for private registry access (default `$WERF_AUTO_PULL_SECRET` or true if not specified)

- `--backend-network`

Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)

- `--bsign-elf-files`

Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).

When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option

- `--build-report-path`

Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`,

`$WERF_CACHE_REPO_2=...`)

- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--copy-from`

Copy file/dir from container to local machine after user command execution. Example:

`"/from/file:to"`. Can be specified multiple times. Can also be defined with `"$WERF_COPY_FROM_*`", e.g. `"WERF_COPY_FROM_1=from:to"`.

- `--copy-to`

Copy file/dir from local machine to container before user command execution. Example:

`"from:/to/file"`. Can be specified multiple times. Can also be defined with `"$WERF_COPY_TO_*`", e.g. `"WERF_COPY_TO_1=from:to"`.

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)

- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read and pull images from the specified repo
- `--dry-run`
Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`
Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`
PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--extra-options`
Pass extra options to "kubectl run" command, which will create a Pod (default \$WERF_EXTRA_OPTIONS)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`
Enable follow mode (default \$WERF_FOLLOW).
The mode allows restarting the command on a new commit.
In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--interactive, -i`
Enable interactive mode (default \$WERF_INTERACTIVE or false if not specified)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)
- `--overrides`
Inline JSON to override/extend any fields in created Pod, e.g. to add imagePullSecrets field (default \$WERF_OVERRIDES). %pod_name%, %container_name%, and %container_image% will be replaced with the names of the created pod, container, and container image, respectively.
- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)

- `--pod`

Set created pod name (default \$WERF_POD or autogenerated if not specified)

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--repo`

Container registry storage address (default \$WERF_REPO)

- `--repo-container-registry`

Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)

- `--repo-docker-hub-token`

repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)

- `--repo-docker-hub-username`

repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)

- `--repo-harbor-password`

repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)

- `--repo-harbor-username`

repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)

- `--repo-quay-token`

repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)

- `--require-built-images, -Z`

Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)

- `--rm`

Remove pod and other created resources after command completion (default \$WERF_RM or true if not specified)

- `--rm-with-namespace`

Remove also a namespace after command completion (default \$WERF_RM_WITH_NAMESPACE or false if not specified)

- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)

- `--sign-cert`

The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)

- `--sign-elf-files`

Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).

When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option

- `--sign-intermediates`

The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)

- `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)

- `--sign-manifest`
Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).
When enabled,
the private signing key must be specified with `--sign-key` option and
the certificate must be specified with `--sign-cert` option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`,
`$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).
Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--tty, -t`
Allocate a TTY (default `$WERF_TTY` or false if not specified)
- `--use-build-report`
Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).
- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state
(`$WERF_VIRTUAL_MERGE` by default)

d8 delivery-kit lint

Usage:

```
d8 delivery-kit lint [IMAGE_NAME...] [options]
```

Lint Helm chart

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: `annoName=annoValue`.
Also, can be specified with `$WERF_ADD_ANNOTATION_*` (e.g.
`$WERF_ADD_ANNOTATION_1=annoName1=annoValue1`,
`$WERF_ADD_ANNOTATION_2=annoName2=annoValue2`)
- `--add-label`
Add label to deploying resources (can specify multiple).

Format: `labelName=labelValue`.

Also, can be specified with `$WERF_ADD_LABEL_*` (e.g.

`$WERF_ADD_LABEL_1=labelName1=labelValue1`,

`$WERF_ADD_LABEL_2=labelName2=labelValue2`)

- `--allow-includes-update`

Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)

- `--backend-network`

Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)

- `--build-report-path`

Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)

- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default `$WERF_DEBUG`).

- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)

- `--delete-propagation`

Set default delete propagation strategy (default `$WERF_DELETE_PROPAGATION` or `Foreground`).

- `--deno-binary-path`

Path to the Deno binary to use instead of auto-downloading (default `$WERF_DENO_BINARY_PATH`)

- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)

- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)

- `--disable-default-secret-values`

Do not use secret values from the default `.helm/secret-values.yaml` file (default `$WERF_DISABLE_DEFAULT_SECRET_VALUES` or false)

- `--disable-default-values`

Do not use values from the default `.helm/values.yaml` file (default

- `$WERF_DISABLE_DEFAULT_VALUES` or false)
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`
Use specified environment (default `$WERF_ENV`)
- `--extra-apiversions`
Extra Kubernetes API versions passed to `$.Capabilities.APIVersions`. Can be also set with `$WERF_EXTRA_APIVERSIONS_*` environment variables, values can be comma-separated
- `--final-images-only`
Process final images only (`$WERF_FINAL_IMAGES_ONLY` or true by default)
- `--final-repo`
Container registry storage address (default `$WERF_FINAL_REPO`)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default `$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD`)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default `$WERF_FINAL_REPO_DOCKER_HUB_TOKEN`)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default `$WERF_FINAL_REPO_DOCKER_HUB_USERNAME`)
- `--final-repo-github-token`
final-repo GitHub token (default `$WERF_FINAL_REPO_GITHUB_TOKEN`)
- `--final-repo-harbor-password`
final-repo Harbor password (default `$WERF_FINAL_REPO_HARBOR_PASSWORD`)
- `--final-repo-harbor-username`
final-repo Harbor username (default `$WERF_FINAL_REPO_HARBOR_USERNAME`)
- `--final-repo-quay-token`
final-repo quay.io token (default `$WERF_FINAL_REPO_QUAY_TOKEN`)
- `--force-adoption`
Always adopt resources, even if they belong to a different Helm release (default `$WERF_FORCE_ADOPTION` or false)
- `--git-work-tree`
Use specified git work tree dir (default `$WERF_WORK_TREE` or lookup for directory that contains `.git` in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default `$WERF_GITERMINISM_CONFIG` or `werf-giterminism.yaml` in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--ignore-bundle-js`
Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default `$WERF_IGNORE_BUNDLE_JS`)
- `--ignore-secret-key`
Disable secrets decryption (default `$WERF_IGNORE_SECRET_KEY`)
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default

\$WERF_INSECURE_HELM_DEPENDENCIES)

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage

- `--introspect-error`

Introspect failed stage in the state, right after running failed assembly instruction

- `--introspect-stage`

Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact
IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name
STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`

Kubernetes API server address (default \$WERF_KUBE_API_SERVER)

- `--kube-auth-password`

Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)

- `--kube-auth-provider`

Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)

- `--kube-auth-provider-config`

Auth provider config for authentication in Kubernetes API (default
\$WERF_KUBE_AUTH_PROVIDER_CONFIG)

- `--kube-auth-username`

Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`

Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)

- `--kube-ca-path`

Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)

- `--kube-cert`

Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)

- `--kube-cert-data`

Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default
\$WERF_KUBE_CERT_DATA)

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or
\$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or
\$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--kube-context-cluster`

Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)

- `--kube-context-user`

- Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables
 - `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)
 - `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)
 - `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)
 - `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)
 - `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
 - `--kube-qps-limit`
Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
 - `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
 - `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
 - `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
 - `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
 - `--kube-version`
Set specific Capabilities.KubeVersion (default \$WERF_KUBE_VERSION)
 - `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
 - `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
 - `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
 - `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
 - `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
 - `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
 - `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
 - `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
 - `--log-verbose`

- Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default `[[ project ]]`-`[[ env ]]` template or `deploy.namespace` custom template from `werf.yaml` or `$WERF_NAMESPACE`)
- `--network-parallelism`
Parallelize some network operations (default `$WERF_NETWORK_PARALLELISM` or 30)
- `--no-final-tracking`
By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default `$WERF_NO_FINAL_TRACKING`)
- `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default `$WERF_NO_REMOVE_MANUAL_CHANGES`)
- `--parallel, -p`
Run in parallel (default `$WERF_PARALLEL` or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default `$WERF_PARALLEL_TASKS_LIMIT` or 5)
- `--platform`
Enable platform emulation when building images with werf, format: `OS/ARCH[/VARIANT]` (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default `$WERF_PROVENANCE_KEYRING`)
- `--provenance-strategy`
Strategy for provenance verifying (default `$WERF_PROVENANCE_STRATEGY`).
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--release`
Use specified Helm release name (default `[[ project ]]`-`[[ env ]]` template or `deploy.helmRelease` custom template from `werf.yaml` or `$WERF_RELEASE`)
- `--release-storage`
How releases should be stored (default `$WERF_RELEASE_STORAGE`)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default `$WERF_RELEASE_STORAGE_SQL_CONNECTION`)
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: `ecr`, `acr`, `default`, `dockerhub`, `gcr`, `github`, `gitlab`, `harbor`, `quay`.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)

- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default `$WERF_REQUIRE_BUILT_IMAGES`)
- `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default `$WERF_RUNTIME_ANNOTATIONS`)
- `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default `$WERF_RUNTIME_LABELS`)
- `--save-build-report`
Save build report (by default `$WERF_SAVE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)
- `--secret-key`
Secret key (default `$WERF_SECRET_KEY`)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with `$WERF_SECRET_VALUES_*` (e.g. `$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml`, `$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml`)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: `key1=val1,key2=val2`).
Also, can be defined with `$WERF_SET_*` (e.g. `$WERF_SET_1=key1=val1`, `$WERF_SET_2=key2=val2`)
- `--set-docker-config-json-value`
Shortcut to set current docker config into the `.Values.dockerconfigjson`
- `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: `key1=path1,key2=path2`).
Also, can be defined with `$WERF_SET_FILE_*` (e.g. `$WERF_SET_FILE_1=key1=path1`, `$WERF_SET_FILE_2=key2=val2`)
- `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: `key1=val1,key2=val2`).
Also, can be defined with `$WERF_SET_JSON_*` (e.g. `$WERF_SET_JSON_1=key1=val1`, `$WERF_SET_JSON_2=key2=val2`)
- `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: `key1=val1,key2=val2`).
Also, can be defined with `$WERF_SET_LITERAL_*` (e.g. `$WERF_SET_LITERAL_1=key1=val1`, `$WERF_SET_LITERAL_2=key2=val2`)
- `--set-root-json`
Set new keys in arbitrary things in the global root context ("`$`"), where the key is the value path

and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: `key1=val1,key2=val2`.

Also, can be defined with `$WERF_SET_ROOT_JSON_*` (e.g. `$WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2`)

- `--set-runtime-json`

Set new keys in `$.Runtime`, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you know what you are doing.

Can specify multiple or separate values with commas: `key1=val1,key2=val2`.

Also, can be defined with `$WERF_SET_RUNTIME_JSON_*` (e.g.

`$WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2`)

- `--set-string`

Set STRING helm values on the command line (can specify multiple or separate values with commas: `key1=val1,key2=val2`).

Also, can be defined with `$WERF_SET_STRING_*` (e.g. `$WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2`)

- `--skip-dependencies-repo-refresh, -L`

Do not refresh helm chart repositories locally cached index

- `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)

- `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default

`$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES`)

- `--skip-tls-verify-kube`

Skip TLS certificate validation when accessing a Kubernetes cluster (default

`$WERF_SKIP_TLS_VERIFY_KUBE`)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default

`$WERF_SKIP_TLS_VERIFY_REGISTRY`)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).

Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`

- `--stub-tags`

Use stubs instead of real tags (default `$WERF_STUB_TAGS`)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--templates-allow-dns`

Allow performing DNS requests in templating (default `$WERF_TEMPLATES_ALLOW_DNS`)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)

- `--use-build-report`

Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED). The alias may contain the following shortcuts:

- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also, can be defined with \$WERF_USE_CUSTOM_TAG (e.g. \$WERF_USE_CUSTOM_TAG="%image%-tag")

- `--validate`

Validate your manifests against the Kubernetes cluster you are currently pointing at (default \$WERF_VALIDATE)

- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with \$WERF_VALUES_* (e.g. \$WERF_VALUES_1=.helm/values_1.yaml, \$WERF_VALUES_2=.helm/values_2.yaml)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit managed-images

Usage:

```
d8 delivery-kit managed-images [options]
```

Work with managed images which will be preserved during cleanup procedure

d8 delivery-kit managed-images add

Usage:

```
d8 delivery-kit managed-images add [options]
```

Add image record to the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in

- working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read and write images to the specified repo
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default

- `$WERF_GITERMINISM_CONFIG` or `werf-giterminism.yaml` in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default `$WERF_INSECURE_REGISTRY`)
- `--kube-config`
Kubernetes config file path (default `$WERF_KUBE_CONFIG`, or `$WERF_KUBECONFIG`, or `$KUBECONFIG`)
- `--kube-config-base64`
Kubernetes config data as base64 string (default `$WERF_KUBE_CONFIG_BASE64` or `$WERF_KUBECONFIG_BASE64` or `$KUBECONFIG_BASE64`)
- `--kube-context`
Kubernetes config context (default `$WERF_KUBE_CONTEXT`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: `OS/ARCH[/VARIANT]` (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

- repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
- repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
- repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
- repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
- repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
- repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
- repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`
- Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`
- Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
- Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization, -S`
- Address of synchronizer for multiple werf processes to work with a single repo.

- Default:
 - \$WERF_SYNCHRONIZATION, or
 - :local if --repo is not specified, or
 - https://delivery-sync.deckhouse.ru if --repo has been specified.

- The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`
- Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
- Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit managed-images ls

Usage:

```
d8 delivery-kit managed-images ls [options]
```

List managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`
- Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`
- Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before

making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)

- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read images from the specified repo
- `--env`
Use specified environment (default `$WERF_ENV`)
- `--final-repo`
Container registry storage address (default `$WERF_FINAL_REPO`)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: `ecr`, `acr`, `default`, `dockerhub`, `gcr`, `github`, `gitlab`, `harbor`, `quay`.
Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or `auto` mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default `$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD`)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default `$WERF_FINAL_REPO_DOCKER_HUB_TOKEN`)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default `$WERF_FINAL_REPO_DOCKER_HUB_USERNAME`)
- `--final-repo-github-token`
final-repo GitHub token (default `$WERF_FINAL_REPO_GITHUB_TOKEN`)
- `--final-repo-harbor-password`
final-repo Harbor password (default `$WERF_FINAL_REPO_HARBOR_PASSWORD`)
- `--final-repo-harbor-username`

- final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).

- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit managed-images rm

Usage:

```
d8 delivery-kit managed-images rm [options]
```

Remove image record from the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`
Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read and write images to the specified repo
- `--env`
Use specified environment (default `$WERF_ENV`)
- `--final-repo`
Container registry storage address (default `$WERF_FINAL_REPO`)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

- final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
- final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
- final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
- final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
- final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
- final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
- final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
- Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
- Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
- Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
- No-op
- `--insecure-registry`
- Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-config`
- Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
- Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
- Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
- Set log color mode.
- Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
- Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
- Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
- Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
- Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
- Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
- Set log terminal width.
- Defaults to:
- * \$WERF_LOG_TERMINAL_WIDTH
- * interactive terminal width or 140
- `--log-time`
- Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--loose-giterminism`

Loose werf giterminism mode restrictions

- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--repo`

Container registry storage address (default \$WERF_REPO)

- `--repo-container-registry`

Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)

- `--repo-docker-hub-token`

repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)

- `--repo-docker-hub-username`

repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)

- `--repo-harbor-password`

repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)

- `--repo-harbor-username`

repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)

- `--repo-quay-token`

repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)

- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local

address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit plan

Usage:

```
d8 delivery-kit plan [options]
```

Prepare deploy plan and show how resources in a Kubernetes cluster would change on next deploy

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: annoName=annoValue.

Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.

\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-custom-tag`

Set tag alias for the content-based tag.

The alias may contain the following shortcuts:

- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);

- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also can be defined with \$WERF_ADD_CUSTOM_TAG_* (e.g.

\$WERF_ADD_CUSTOM_TAG_1="%image%-tag1", \$WERF_ADD_CUSTOM_TAG_2="%image%-tag2")

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.

Also, can be specified with \$WERF_ADD_LABEL_* (e.g.

\$WERF_ADD_LABEL_1=labelName1=labelValue1,

\$WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--allowed-backend-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)

- `--allowed-docker-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or

- `$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`

 - `--allowed-local-cache-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of local cache (`~/.werf/local_cache` by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`)
 - `--allowed-local-cache-volume-usage-margin`

During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or `$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`)
 - `--annotate-layers-with-dm-verity-root-hash`

Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
 - `--backend-network`

Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
 - `--backend-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_BACKEND_STORAGE_PATH`)
 - `--bsign-elf-files`

Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
 - `--build-report-path`

Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or `".werf-build-report.json"` if not set). Extension must be either `.json` for JSON format or `.env` for env-file format. If extension not specified, then `.json` is used
 - `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
 - `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
 - `--config-render-path`

Custom path for storing rendered configuration file
 - `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
 - `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io
 - `--debug`

Enable debug (default `$WERF_DEBUG`).
 - `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)
 - `--delete-propagation`

Set default delete propagation strategy (default `$WERF_DELETE_PROPAGATION` or `Foreground`).
 - `--deno-binary-path`

Path to the Deno binary to use instead of auto-downloading (default `$WERF_DENO_BINARY_PATH`)
 - `--deploy-graph-path`

Save deploy graph path to the specified file (by default `$WERF_DEPLOY_GRAPH_PATH`). Extension must be `.dot` or not specified. If extension not specified, then `.dot` is used
 - `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--diff-context-lines`

Show N lines of context around diffs (\$WERF_DIFF_CONTEXT_LINES by default)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)

- `--disable-default-secret-values`

Do not use secret values from the default .helm/secret-values.yaml file (default \$WERF_DISABLE_DEFAULT_SECRET_VALUES or false)

- `--disable-default-values`

Do not use values from the default .helm/values.yaml file (default \$WERF_DISABLE_DEFAULT_VALUES or false)

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo, to pull base images

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)

- `--elf-pgp-private-key-base64`

Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)

- `--elf-pgp-private-key-fingerprint`

PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)

- `--elf-pgp-private-key-passphrase`

Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--exit-code`

Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default \$WERF_EXIT_CODE or false)

- `--final-images-only`

Process final images only (\$WERF_FINAL_IMAGES_ONLY or true by default)

- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

- final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
- final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
- final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
- final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
- final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
- final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
- final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`
- Enable follow mode (default \$WERF_FOLLOW).
- The mode allows restarting the command on a new commit.
- In development mode (`--dev`), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--force-adoption`
- Always adopt resources, even if they belong to a different Helm release (default \$WERF_FORCE_ADOPTION or false)
- `--git-work-tree`
- Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
- Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
- Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`
- No-op
- `--ignore-bundle-js`
- Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default \$WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`
- Disable secrets decryption (default \$WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`
- Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`
- Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--introspect-before-error`
- Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`
- Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`
- Introspect a specific stage. The option can be used multiple times to introspect several stages.

- There are the following formats to use:
- * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
- * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`

Kubernetes API server address (default \$WERF_KUBE_API_SERVER)

- `--kube-auth-password`

Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)

- `--kube-auth-provider`

Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)

- `--kube-auth-provider-config`

Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)

- `--kube-auth-username`

Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`

Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)

- `--kube-ca-path`

Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)

- `--kube-cert`

Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)

- `--kube-cert-data`

Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--kube-context-cluster`

Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)

- `--kube-context-user`

Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)

- `--kube-impersonate-group`

Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables

- `--kube-impersonate-uid`

Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)

- `--kube-impersonate-user`

Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)

- `--kube-key`

Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)

- `--kube-key-data`

Pass PEM-encoded TLS client key for connecting to Kubernetes API (default

- `$WERF_KUBE_KEY_DATA`
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default `$WERF_KUBE_PROXY_URL`)
- `--kube-qps-limit`
Kubernetes client QPS limit (default `$WERF_KUBE_QPS_LIMIT` or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default `$WERF_KUBE_REQUEST_TIMEOUT`)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default `$WERF_KUBE_TLS_SERVER`)
- `--kube-token`
Kubernetes bearer token used for authentication (default `$WERF_KUBE_TOKEN`)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default `$WERF_KUBE_TOKEN_PATH`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default `[[ project ]]`-`[[ env ]]` template or `deploy.namespace` custom template from `werf.yaml` or `$WERF_NAMESPACE`)
- `--network-parallelism`
Parallelize some network operations (default `$WERF_NETWORK_PARALLELISM` or 30)
- `--no-final-tracking`
By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default `$WERF_NO_FINAL_TRACKING`)
- `--no-install-crds`
Do not install CRDs from "crds/" directories of installed charts (default `$WERF_NO_INSTALL_CRDS`)
- `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the

- manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default \$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
Strategy for provenance verifying (default \$WERF_PROVENANCE_STRATEGY).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or \$WERF_RELEASE)
- `--release-info-annotations`
Add annotations to release metadata (default \$WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`
Add Helm release labels (can specify multiple). Kind of labels depends on release storage driver. Format: labelName=labelValue.
Also, can be specified with \$WERF_RELEASE_LABEL_* (e.g. \$WERF_RELEASE_LABEL_1=labelName1=labelValue1, \$WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`
Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

- repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)
- `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default \$WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default \$WERF_RUNTIME_LABELS)
- `--save-build-report`
Save build report (by default \$WERF_SAVE_BUILD_REPORT or false). Its path and format configured with `--build-report-path`
- `--save-plan`
Save the gzip-compressed JSON install plan to the specified file
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--secret-key`
Secret key (default \$WERF_SECRET_KEY)
- `--secret-values`
Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2). Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`
Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`
Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2). Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1, \$WERF_SET_FILE_2=key2=val2)
- `--set-json`
Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2). Also, can be defined with \$WERF_SET_JSON_* (e.g. \$WERF_SET_JSON_1=key1=val1, \$WERF_SET_JSON_2=key2=val2)
- `--set-literal`
Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2). Also, can be defined with \$WERF_SET_LITERAL_* (e.g. \$WERF_SET_LITERAL_1=key1=val1, \$WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`
Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you REALLY know what you are doing. Can specify multiple or separate values with

- commas: key1=val1,key2=val2.
- Also, can be defined with \$WERF_SET_ROOT_JSON_* (e.g. \$WERF_SET_ROOT_JSON_1=key1=val1, \$WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`
Set new keys in \$.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.
Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with \$WERF_SET_RUNTIME_JSON_* (e.g. \$WERF_SET_RUNTIME_JSON_1=key1=val1, \$WERF_SET_RUNTIME_JSON_2=key2=val2)
 - `--set-string`
Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Also, can be defined with \$WERF_SET_STRING_* (e.g. \$WERF_SET_STRING_1=key1=val1, \$WERF_SET_STRING_2=key2=val2)
 - `--show-insignificant-diffs`
Show insignificant diff lines (\$WERF_SHOW_INSIGNIFICANT_DIFFS by default)
 - `--show-plan`
Show plan artifact planned changes
 - `--show-sensitive-diffs`
Show sensitive diff lines (\$WERF_SHOW_SENSITIVE_DIFFS by default)
 - `--show-verbose-crd-diffs`
Show verbose CRD diff lines (\$WERF_SHOW_VERBOSE_CRD_DIFFS by default)
 - `--show-verbose-diffs`
Show verbose diff lines (\$WERF_SHOW_VERBOSE_DIFFS by default)
 - `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)
 - `--sign-elf-files`
Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).
When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
 - `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)
 - `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)
 - `--sign-manifest`
Enable image manifest signing (default \$WERF_SIGN_MANIFEST).
When enabled,
the private signing key must be specified with --sign-key option and
the certificate must be specified with --sign-cert option
 - `--skip-dependencies-repo-refresh, -L`
Do not refresh helm chart repositories locally cached index
 - `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)
 - `--skip-tls-verify-helm-dependencies`
Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
 - `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
 - `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default

`$WERF_SKIP_TLS_VERIFY_REGISTRY)`

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with `$WERF_SSH_KEY_*` (e.g. `$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa`, `$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa`).

Defaults to `$WERF_SSH_KEY_*`, system ssh-agent or `~/.ssh/{id_rsa|id_dsa}`

- `--status-progress-period`

No-op

- `--stub-tags`

Use stubs instead of real tags (default `$WERF_STUB_TAGS`)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--templates-allow-dns`

Allow performing DNS requests in templating (default `$WERF_TEMPLATES_ALLOW_DNS`)

- `--timeout, -t`

No-op

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)

- `--use-build-report`

Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).

The alias may contain the following shortcuts:

- `%image%`, `%image_slug%` or `%image_safe_slug%` to use the image name (necessary if there is more than one image in the werf config);
- `%image_content_based_tag%` to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also, can be defined with `$WERF_USE_CUSTOM_TAG` (e.g. `$WERF_USE_CUSTOM_TAG="%image%-tag"`)

- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with `$WERF_VALUES_*` (e.g. `$WERF_VALUES_1=.helm/values_1.yaml`, `$WERF_VALUES_2=.helm/values_2.yaml`)

- `--verbose`

Enable verbose output (default `$WERF_VERBOSE`).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (`$WERF_VIRTUAL_MERGE` by default)

- `--without-images`

Disable building of images defined in the `werf.yaml` (if any) and usage of such images in the `.helm/templates` (`$WERF_WITHOUT_IMAGES` or false by default — e.g. enable all images defined in the `werf.yaml` by default)

d8 delivery-kit purge

Usage:

```
d8 delivery-kit purge [options]
```

Purge all project images in the container registry

Options

- `--allow-includes-update`
Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`
During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)
- `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io

- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--debug-templates`
Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)
- `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to delete images from the specified repo
- `--docker-server-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`
Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

- final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`
No-op
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * \$WERF_LOG_TERMINAL_WIDTH
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit render

Usage:

```
d8 delivery-kit render [IMAGE_NAME...] [options]
```

Render Kubernetes templates

Options

- `--add-annotation`
Add annotation to deploying resources (can specify multiple).
Format: annoName=annoValue.
Also, can be specified with `$WERF_ADD_ANNOTATION_*` (e.g. `$WERF_ADD_ANNOTATION_1=annoName1=annoValue1`, `$WERF_ADD_ANNOTATION_2=annoName2=annoValue2`)
- `--add-label`
Add label to deploying resources (can specify multiple).
Format: labelName=labelValue.
Also, can be specified with `$WERF_ADD_LABEL_*` (e.g. `$WERF_ADD_LABEL_1=labelName1=labelValue1`, `$WERF_ADD_LABEL_2=labelName2=labelValue2`)
- `--allow-includes-update`
Allow use includes latest versions (default `$WERF_ALLOW_INCLUDES_UPDATE` or false)
- `--annotate-layers-with-dm-verity-root-hash`
Enable annotation of image layers with dm-verity root hash (default `$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH`)
- `--backend-network`
Network mode for the build containers (`$WERF_BACKEND_NETWORK` or nothing by default)
- `--bsign-elf-files`
Enable ELF files signing with bsign (default `$WERF_BSIGN_ELF_FILES`).
When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option
- `--build-report-path`
Change build report path and format (by default `$WERF_BUILD_REPORT_PATH` or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`
Use custom configuration file (default `$WERF_CONFIG` or werf.yaml in the project directory)
- `--config-render-path`
Custom path for storing rendered configuration file
- `--config-templates-dir`
Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or .werf in working directory)
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`
Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or false)
- `--deno-binary-path`
Path to the Deno binary to use instead of auto-downloading (default `$WERF_DENO_BINARY_PATH`)
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

- Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
- `--disable-default-secret-values`
Do not use secret values from the default .helm/secret-values.yaml file (default \$WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`
Do not use values from the default .helm/values.yaml file (default \$WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--elf-gpg-private-key-base64`
Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-gpg-private-key-fingerprint`
PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-gpg-private-key-passphrase`
Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`
Use specified environment (default \$WERF_ENV)
- `--extra-apiversions`
Extra Kubernetes API versions passed to \$.Capabilities.APIVersions. Can be also set with \$WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--final-images-only`
Process final images only (\$WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`
Container registry storage address (default \$WERF_FINAL_REPO)
- `--final-repo-container-registry`
Choose final-repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

- final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--force-adoption`
Always adopt resources, even if they belong to a different Helm release (default \$WERF_FORCE_ADOPTION or false)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`
No-op
- `--ignore-bundle-js`
Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default \$WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`
Disable secrets decryption (default \$WERF_IGNORE_SECRET_KEY)
- `--include-crds`
Include CRDs in the templated output (default \$WERF_INCLUDE_CRDS)
- `--insecure-helm-dependencies`
Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default \$WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--introspect-before-error`
Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`
Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`
Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact
IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name
STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`
Kubernetes API server address (default \$WERF_KUBE_API_SERVER)
- `--kube-auth-password`
Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`
Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`
Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

- Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`
Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)
- `--kube-ca-path`
Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`
Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
- `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
- `--kube-version`

- Set specific Capabilities.KubeVersion (default \$WERF_KUBE_VERSION)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)
- `--network-parallelism`
Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)
- `--output`
Write render output to the specified file instead of stdout (\$WERF_RENDER_OUTPUT by default)
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--provenance-keyring`
Path to keyring containing public keys to verify chart provenance (default \$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
Strategy for provenance verifying (default \$WERF_PROVENANCE_STRATEGY).
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--release`
Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or \$WERF_RELEASE)
- `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)

- `--releases-history-max`

Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)

- `--repo`

Container registry storage address (default \$WERF_REPO)

- `--repo-container-registry`

Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)

- `--repo-docker-hub-token`

repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)

- `--repo-docker-hub-username`

repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)

- `--repo-harbor-password`

repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)

- `--repo-harbor-username`

repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)

- `--repo-quay-token`

repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)

- `--require-built-images, -Z`

Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)

- `--save-build-report`

Save build report (by default \$WERF_SAVE_BUILD_REPORT or false). Its path and format configured with `--build-report-path`

- `--secondary-repo`

Specify one or multiple secondary read-only repos with images that will be used as a cache.

Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)

- `--secret-key`

Secret key (default \$WERF_SECRET_KEY)

- `--secret-values`

Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with \$WERF_SECRET_VALUES_* (e.g. \$WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, \$WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)

- `--set`

Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).

Also, can be defined with \$WERF_SET_* (e.g. \$WERF_SET_1=key1=val1, \$WERF_SET_2=key2=val2)

- `--set-docker-config-json-value`

Shortcut to set current docker config into the .Values.dockerconfigjson

- `--set-file`

Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).

Also, can be defined with \$WERF_SET_FILE_* (e.g. \$WERF_SET_FILE_1=key1=path1,

- `$WERF_SET_FILE_2=key2=val2)`

 - `--set-json`

Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: `key1=val1,key2=val2`).

Also, can be defined with `$WERF_SET_JSON_*` (e.g. `$WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2`)
 - `--set-literal`

Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: `key1=val1,key2=val2`).

Also, can be defined with `$WERF_SET_LITERAL_*` (e.g. `$WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2`)
 - `--set-root-json`

Set new keys in arbitrary things in the global root context ("\$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: `key1=val1,key2=val2`.

Also, can be defined with `$WERF_SET_ROOT_JSON_*` (e.g. `$WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2`)
 - `--set-runtime-json`

Set new keys in `$.Runtime`, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use `--set-json` instead, unless you know what you are doing. Can specify multiple or separate values with commas: `key1=val1,key2=val2`.

Also, can be defined with `$WERF_SET_RUNTIME_JSON_*` (e.g. `$WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2`)
 - `--set-string`

Set STRING helm values on the command line (can specify multiple or separate values with commas: `key1=val1,key2=val2`).

Also, can be defined with `$WERF_SET_STRING_*` (e.g. `$WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2`)
 - `--show-only, -s`

only show manifests rendered from the given templates
 - `--sign-cert`

The leaf certificate as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_CERT`)
 - `--sign-elf-files`

Enable ELF files signing (default `$WERF_SIGN_ELF_FILES`).

When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option
 - `--sign-intermediates`

The intermediate certificates as path to PEM file or base64-encoded PEM (default `$WERF_SIGN_INTERMEDIATES`)
 - `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default `$WERF_SIGN_KEY`)
 - `--sign-manifest`

Enable image manifest signing (default `$WERF_SIGN_MANIFEST`).

When enabled, the private signing key must be specified with `--sign-key` option and the certificate must be specified with `--sign-cert` option
 - `--skip-dependencies-repo-refresh, -L`

Do not refresh helm chart repositories locally cached index
 - `--skip-image-spec-stage`

Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
 - `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default \$WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)

- `--skip-tls-verify-kube`

Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--status-progress-period`

No-op

- `--stub-tags`

Use stubs instead of real tags (default \$WERF_STUB_TAGS)

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--templates-allow-dns`

Allow performing DNS requests in templating (default \$WERF_TEMPLATES_ALLOW_DNS)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

Use build report, previously saved with --save-build-report (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).

The alias may contain the following shortcuts:

- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.

For cleaning custom tags and associated content-based tag are treated as one.

Also, can be defined with \$WERF_USE_CUSTOM_TAG (e.g. \$WERF_USE_CUSTOM_TAG="%image%-tag")

- `--validate`

Validate your manifests against the Kubernetes cluster you are currently pointing at (default \$WERF_VALIDATE)

- `--values`

Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with \$WERF_VALUES_* (e.g. \$WERF_VALUES_1=.helm/values_1.yaml, \$WERF_VALUES_2=.helm/values_2.yaml)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state

(\$WERF_VIRTUAL_MERGE by default)

- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates (\$WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit rollback

Usage:

```
d8 delivery-kit rollback [options]
```

Rollback the Helm release

Options

- `--add-annotation`

Add annotation to deploying resources (can specify multiple).

Format: annoName=annoValue.

Also, can be specified with \$WERF_ADD_ANNOTATION_* (e.g.

\$WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

\$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-label`

Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.

Also, can be specified with \$WERF_ADD_LABEL_* (e.g.

\$WERF_ADD_LABEL_1=labelName1=labelValue1,

\$WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--delete-propagation`

Set default delete propagation strategy (default \$WERF_DELETE_PROPAGATION or Foreground).

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go,

\$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--force-adoption`
Always adopt resources, even if they belong to a different Helm release (default \$WERF_FORCE_ADOPTION or false)
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`
No-op
- `--kube-api-server`
Kubernetes API server address (default \$WERF_KUBE_API_SERVER)
- `--kube-auth-password`
Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`
Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`
Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`
Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`
Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)
- `--kube-ca-path`
Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)
- `--kube-cert`
Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)
- `--kube-cert-data`
Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)
- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`
Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)

- `--kube-impersonate-user`
Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)
- `--kube-key-data`
Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`
Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`
Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
- `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--namespace`
Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or \$WERF_NAMESPACE)
- `--network-parallelism`
Parallelize some network operations (default \$WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

- By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default \$WERF_NO_FINAL_TRACKING)
- `--no-notes`
Don't show release notes at the end of the release (default \$WERF_NO_NOTES)
 - `--no-pod-logs`
Disable Pod logs collection and printing (default \$WERF_NO_POD_LOGS or false)
 - `--no-remove-manual-changes`
Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default \$WERF_NO_REMOVE_MANUAL_CHANGES)
 - `--quiet`
Disable explanatory output (default \$WERF_QUIET).
 - `--release`
Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or \$WERF_RELEASE)
 - `--release-info-annotations`
Add annotations to release metadata (default \$WERF_RELEASE_INFO_ANNOTATIONS)
 - `--release-label`
Add Helm release labels (can specify multiple). Kind of labels depends on release storage driver. Format: `labelName=labelValue`.
Also, can be specified with `$WERF_RELEASE_LABEL_*` (e.g. `$WERF_RELEASE_LABEL_1=labelName1=labelValue1`, `$WERF_RELEASE_LABEL_2=labelName2=labelValue2`)
 - `--release-storage`
How releases should be stored (default \$WERF_RELEASE_STORAGE)
 - `--release-storage-sql-connection`
SQL Connection String for Helm SQL Storage (default \$WERF_RELEASE_STORAGE_SQL_CONNECTION)
 - `--releases-history-max`
Max releases to keep in release storage (\$WERF_RELEASES_HISTORY_MAX or 5 by default)
 - `--revision`
Revision number to rollback to (if not specified, rolls back to previous revision)
 - `--rollback-graph-path`
Save rollback graph path to the specified file (by default \$WERF_ROLLBACK_GRAPH_PATH). Extension must be `.dot` or not specified. If extension not specified, then `.dot` is used
 - `--rollback-report-path`
Change rollback report path and format (by default \$WERF_ROLLBACK_REPORT_PATH or `".werf-rollback-report.json"` if not set). Extension must be `.json` for JSON format. If extension not specified, then `.json` is used
 - `--runtime-annotations`
Add annotations which will not trigger resource updates to all resources (default \$WERF_RUNTIME_ANNOTATIONS)
 - `--runtime-labels`
Add labels which will not trigger resource updates to all resources (default \$WERF_RUNTIME_LABELS)
 - `--save-rollback-report`
Save rollback report (by default \$WERF_SAVE_ROLLBACK_REPORT or false). Its path and format configured with `--rollback-report-path`
 - `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
 - `--status-progress-period`
Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

\$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--timeout, -t`

Resources tracking timeout in seconds (\$WERF_TIMEOUT by default)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit run

Usage:

```
d8 delivery-kit run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image

Options

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--annotate-layers-with-dm-verity-root-hash`

Enable annotation of image layers with dm-verity root hash (default

\$WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)

- `--backend-network`

Network mode for the build containers (\$WERF_BACKEND_NETWORK or nothing by default)

- `--bash`

Use predefined docker options and command for debug

- `--bsign-elf-files`

Enable ELF files signing with bsign (default \$WERF_BSIGN_ELF_FILES).

When enabled, the private elf key must be specified with `--elf-pgp-private-key-base64` or `--elf-pgp-private-key-fingerprint` option

- `--build-report-path`

Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used

- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)

- `--config`

Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)

- `--config-render-path`

Custom path for storing rendered configuration file

- `--config-templates-dir`

Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--debug-templates`

Enable debug mode for Go templates (default \$WERF_DEBUG_TEMPLATES or false)

- `--dev`

Enable development mode (default \$WERF_DEV).

The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)

- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read and pull images from the specified repo

- `--docker-options`

Define docker run options (default \$WERF_DOCKER_OPTIONS)

- `--dry-run`

Indicate what the command would do without actually doing that (default \$WERF_DRY_RUN)

- `--elf-pgp-private-key-base64`

Base64-encoded PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_BASE64)

- `--elf-pgp-private-key-fingerprint`

PGP private key fingerprint (default \$WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)

- `--elf-pgp-private-key-passphrase`

Passphrase for the PGP private key (default \$WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)

- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)

- `--follow`

Enable follow mode (default \$WERF_FOLLOW).

The mode allows restarting the command on a new commit.

- In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
 - `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
 - `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
 - `--insecure-helm-dependencies`
No-op
 - `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
 - `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
 - `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
 - `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
 - `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
 - `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
 - `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
 - `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
 - `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
 - `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
 - `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
 - `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
 - `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
 - `--loose-giterminism`
Loose werf giterminism mode restrictions
 - `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
 - `--quiet`
Disable explanatory output (default \$WERF_QUIET).
 - `--repo`

- Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--require-built-images, -Z`
Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default \$WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache.
Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--shell`
Use predefined docker options and command for debug
- `--sign-cert`
The leaf certificate as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_CERT)
- `--sign-elf-files`
Enable ELF files signing (default \$WERF_SIGN_ELF_FILES).
When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`
The intermediate certificates as path to PEM file or base64-encoded PEM (default \$WERF_SIGN_INTERMEDIATES)
- `--sign-key`
The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default \$WERF_SIGN_KEY)
- `--sign-manifest`
Enable image manifest signing (default \$WERF_SIGN_MANIFEST).
When enabled,
the private signing key must be specified with --sign-key option and
the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

Use only specific ssh key(s).

Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).

Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--synchronization, -S`

Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--use-build-report`

Use build report, previously saved with --save-build-report (by default \$WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`

Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom

Usage:

```
d8 delivery-kit sbom [options]
```

Work with werf SBOM images

d8 delivery-kit sbom get

Usage:

```
d8 delivery-kit sbom get [IMAGE_NAME] [options]
```

Get SBOM of an image

Options

- `--allowed-backend-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)

- `--allowed-docker-storage-volume-usage-margin`

During cleanup of least recently used local backend (Docker or Buildah) images werf would delete

- images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`
Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
 - `--allowed-local-cache-volume-usage-margin`
During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or \$WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
 - `--backend-storage-path`
Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)
 - `--build-report-path`
Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
 - `--cache-repo`
Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with \$WERF_CACHE_REPO_* (e.g. \$WERF_CACHE_REPO_1=..., \$WERF_CACHE_REPO_2=...)
 - `--config`
Use custom configuration file (default \$WERF_CONFIG or werf.yaml in the project directory)
 - `--config-templates-dir`
Custom configuration templates directory (default \$WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
 - `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
 - `--debug`
Enable debug (default \$WERF_DEBUG).
 - `--dev`
Enable development mode (default \$WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development
 - `--dev-branch`
Set dev git branch name (default \$WERF_DEV_BRANCH or "_werf-dev")
 - `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
Also, can be specified with \$WERF_DEV_IGNORE_* (e.g. \$WERF_DEV_IGNORE_TESTS=*_test.go, \$WERF_DEV_IGNORE_DOCS=path/to/docs)
 - `--digest`
Digest of the image to get SBOM for (mutually exclusive with --tag)
 - `--dir`
Use specified project directory where project's werf.yaml and other configuration files should reside (default \$WERF_DIR or current working directory)
 - `--disable-auto-host-cleanup`
Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
 - `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo and to pull base images

- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_DOCKER_SERVER_STORAGE_PATH)

- `--env`

Use specified environment (default \$WERF_ENV)

- `--final-repo`

Container registry storage address (default \$WERF_FINAL_REPO)

- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)

- `--final-repo-docker-hub-username`

final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)

- `--final-repo-github-token`

final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)

- `--final-repo-harbor-password`

final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)

- `--final-repo-harbor-username`

final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--introspect-before-error`

Introspect failed stage in the clean state, before running all assembly instructions of the stage

- `--introspect-error`

Introspect failed stage in the state, right after running failed assembly instruction

- `--introspect-stage`

Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image

specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`

Kubernetes API server address (default \$WERF_KUBE_API_SERVER)

- `--kube-auth-password`

Basic auth password for Kubernetes API (default \$WERF_KUBE_AUTH_PASSWORD)

- `--kube-auth-provider`

Auth provider name for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER)

- `--kube-auth-provider-config`

Auth provider config for authentication in Kubernetes API (default \$WERF_KUBE_AUTH_PROVIDER_CONFIG)

- `--kube-auth-username`

Basic auth username for Kubernetes API (default \$WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

Kubernetes client burst limit (default \$WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`

Pass Kubernetes API server TLS CA data (default \$WERF_KUBE_CA_DATA)

- `--kube-ca-path`

Kubernetes API server CA path (default \$WERF_KUBE_CA_PATH)

- `--kube-cert`

Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT)

- `--kube-cert-data`

Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default \$WERF_KUBE_CERT_DATA)

- `--kube-config`

Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)

- `--kube-config-base64`

Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)

- `--kube-context`

Kubernetes config context (default \$WERF_KUBE_CONTEXT)

- `--kube-context-cluster`

Use cluster from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_CLUSTER)

- `--kube-context-user`

Use user from Kubeconfig for current context (default \$WERF_KUBE_CONTEXT_USER)

- `--kube-impersonate-group`

Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with \$WERF_KUBE_IMPERSONATE_GROUP_* environment variables

- `--kube-impersonate-uid`

Sets Impersonate-Uid header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_UID)

- `--kube-impersonate-user`

Sets Impersonate-User header when authenticating in Kubernetes (default \$WERF_KUBE_IMPERSONATE_USER)

- `--kube-key`

Path to PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY)

- `--kube-key-data`

Pass PEM-encoded TLS client key for connecting to Kubernetes API (default \$WERF_KUBE_KEY_DATA)

- `--kube-proxy-url`

- Proxy URL to use for proxying all requests to Kubernetes API (default \$WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`
Kubernetes client QPS limit (default \$WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`
Timeout for all requests to Kubernetes API (default \$WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`
Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default \$WERF_KUBE_TLS_SERVER)
- `--kube-token`
Kubernetes bearer token used for authentication (default \$WERF_KUBE_TOKEN)
- `--kube-token-path`
Path to file with bearer token for authentication in Kubernetes (default \$WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-project-dir`
Print current project directory path (default \$WERF_LOG_PROJECT_DIR)
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT]
(\$WERF_PLATFORM or \$DOCKER_DEFAULT_PLATFORM by default)
- `--project-name, -N`
Set a specific project name (default \$WERF_PROJECT_NAME)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.

Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--save-build-report`
Save build report (by default \$WERF_SAVE_BUILD_REPORT or false). Its path and format configured with `--build-report-path`
- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with \$WERF_SECONDARY_REPO_* (e.g. \$WERF_SECONDARY_REPO_1=..., \$WERF_SECONDARY_REPO_2=...)
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default \$WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-kube`
Skip TLS certificate validation when accessing a Kubernetes cluster (default \$WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
Use only specific ssh key(s).
Can be specified with \$WERF_SSH_KEY_* (e.g. \$WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, \$WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to \$WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- \$WERF_SYNCHRONIZATION, or
- :local if `--repo` is not specified, or
- <https://delivery-sync.deckhouse.ru> if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tag`
Content-based tag of the image to get SBOM for (mutually exclusive with `--digest`)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

- `--virtual-merge`
Enable virtual/ephemeral merge commit mode when building current application state (\$WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom merge

Usage:

```
d8 delivery-kit sbom merge [options]
```

Merge per-image SBOMs into a product-level SBOM

Options

- `--app-name`
Application/product name for the merged SBOM metadata
- `--app-version`
Application/product version for the merged SBOM metadata
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--docker-config`
Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to pull SBOM images from the specified repo
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--input`
Path to JSON mapping file (image name -> sha256:digest)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--ispras-format`
ISPRAS SBOM format: "oss" or "container"
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--manufacturer`

- `--output, -o`
Output file path (defaults to stdout)
- `--parallel, -p`
Run in parallel (default \$WERF_PARALLEL or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--repo`
Container registry storage address (default \$WERF_REPO)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default \$WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default \$WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
repo Docker Hub token (default \$WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
repo Docker Hub username (default \$WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
repo GitHub token (default \$WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
repo Harbor password (default \$WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
repo Harbor username (default \$WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
repo quay.io token (default \$WERF_REPO_QUAY_TOKEN)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit sbom validate

Usage:

```
d8 delivery-kit sbom validate [options]
```

Validate SBOM file against ISPRAS schemas

Options

- `--check-vcs`
Enable VCS URL validation
- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default \$WERF_DEBUG).
- `--docker-config`

Specify docker config directory path. Default \$WERF_DOCKER_CONFIG or \$DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to pull the ISPRAS SBOM checker image

- `--home-dir`

Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)

- `--insecure-registry`

Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)

- `--ispras-format`

ISPRAS SBOM format: oss or container

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--path`

Path to CycloneDX JSON SBOM file (repeatable)

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--skip-tls-verify-registry`

Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--tmp-dir`

Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit slugify

Usage:

```
d8 delivery-kit slugify STRING [options]
```

Print slugged string by specified format.

Options

- `--debug`

Enable debug (default \$WERF_DEBUG).

- `--format, -f`

r|helm-release: suitable for Helm Release

ns|kubernetes-namespace: suitable for Kubernetes Namespace

tag|docker-tag: suitable for Docker Tag

- `--log-color-mode`

Set log color mode.

Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.

Default \$WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

Enable debug (default \$WERF_LOG_DEBUG).

- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-quiet`

Disable explanatory output (default \$WERF_LOG_QUIET).

- `--log-terminal-width`

Set log terminal width.

Defaults to:

* \$WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140

- `--log-time`

Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).

- `--log-time-format`

Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

Enable verbose output (default \$WERF_LOG_VERBOSE).

- `--quiet`

Disable explanatory output (default \$WERF_QUIET).

- `--verbose`

Enable verbose output (default \$WERF_VERBOSE).

d8 delivery-kit stages

Usage:

```
d8 delivery-kit stages [options]
```

Work with werf stages

d8 delivery-kit stages copy

Usage:

```
d8 delivery-kit stages copy [options]
```

Copy stages between container registry and archive

Options

- `--all`

Copy all project stages (default: true). Use --all=false to copy stages for current build only. Note: this flag is ignored when copying from archive to container registry.

- `--allow-includes-update`

Allow use includes latest versions (default \$WERF_ALLOW_INCLUDES_UPDATE or false)

- `--backend-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use \$WERF_BACKEND_STORAGE_PATH)

- `--build-report-path`

Change build report path and format (by default \$WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If

- extension not specified, then .json is used
- `--cache-repo`

Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.

Also, can be specified with `$WERF_CACHE_REPO_*` (e.g. `$WERF_CACHE_REPO_1=...`, `$WERF_CACHE_REPO_2=...`)
- `--config`

Use custom configuration file (default `$WERF_CONFIG` or `werf.yaml` in the project directory)
- `--config-render-path`

Custom path for storing rendered configuration file
- `--config-templates-dir`

Custom configuration templates directory (default `$WERF_CONFIG_TEMPLATES_DIR` or `.werf` in working directory)
- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io
- `--debug`

Enable debug (default `$WERF_DEBUG`).
- `--debug-templates`

Enable debug mode for Go templates (default `$WERF_DEBUG_TEMPLATES` or `false`)
- `--dev`

Enable development mode (default `$WERF_DEV`).

The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`

Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--dir`

Use specified project directory where project's `werf.yaml` and other configuration files should reside (default `$WERF_DIR` or current working directory)
- `--docker-config`

Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repos
- `--docker-server-storage-path`

Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use `$WERF_DOCKER_SERVER_STORAGE_PATH`)
- `--env`

Use specified environment (default `$WERF_ENV`)
- `--final-images-only`

Process final images only (`$WERF_FINAL_IMAGES_ONLY` or `false` by default)
- `--final-repo`

Container registry storage address (default `$WERF_FINAL_REPO`)
- `--final-repo-container-registry`

Choose final-repo container registry implementation.

The following container registries are supported: `ecr`, `acr`, `default`, `dockerhub`, `gcr`, `github`, `gitlab`, `harbor`, `quay`.

Default `$WERF_FINAL_REPO_CONTAINER_REGISTRY` or `auto` mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`
final-repo Docker Hub password (default \$WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
final-repo Docker Hub token (default \$WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
final-repo Docker Hub username (default \$WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
final-repo GitHub token (default \$WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
final-repo Harbor password (default \$WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
final-repo Harbor username (default \$WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
final-repo quay.io token (default \$WERF_FINAL_REPO_QUAY_TOKEN)
- `--from`
Source address to copy stages from. Use archive:PATH for stage archive or [docker:///]REPO for container registry.
- `--git-work-tree`
Use specified git work tree dir (default \$WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
Custom path to the giterminism configuration file relative to working directory (default \$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default \$WERF_HOME or ~/.werf)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default \$WERF_INSECURE_REGISTRY)
- `--introspect-before-error`
Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`
Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`
Introspect a specific stage. The option can be used multiple times to introspect several stages.

There are the following formats to use:

* specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME

* specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.

STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).

- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
 - * `$WERF_LOG_TERMINAL_WIDTH`
 - * interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--parallel, -p`
Run in parallel (default `$WERF_PARALLEL` or true)
- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default `$WERF_PARALLEL_TASKS_LIMIT` or 5)
- `--platform`
Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] (`$WERF_PLATFORM` or `$DOCKER_DEFAULT_PLATFORM` by default)
- `--project-name, -N`
Set a specific project name (default `$WERF_PROJECT_NAME`)
- `--quiet`
Disable explanatory output (default `$WERF_QUIET`).
- `--repo`
Container registry storage address (default `$WERF_REPO`)
- `--repo-container-registry`
Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default `$WERF_REPO_CONTAINER_REGISTRY` or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
repo Docker Hub password (default `$WERF_REPO_DOCKER_HUB_PASSWORD`)
- `--repo-docker-hub-token`
repo Docker Hub token (default `$WERF_REPO_DOCKER_HUB_TOKEN`)
- `--repo-docker-hub-username`
repo Docker Hub username (default `$WERF_REPO_DOCKER_HUB_USERNAME`)
- `--repo-github-token`
repo GitHub token (default `$WERF_REPO_GITHUB_TOKEN`)
- `--repo-harbor-password`
repo Harbor password (default `$WERF_REPO_HARBOR_PASSWORD`)
- `--repo-harbor-username`
repo Harbor username (default `$WERF_REPO_HARBOR_USERNAME`)
- `--repo-quay-token`
repo quay.io token (default `$WERF_REPO_QUAY_TOKEN`)
- `--save-build-report`
Save build report (by default `$WERF_SAVE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`

- `--secondary-repo`
Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with `$WERF_SECONDARY_REPO_*` (e.g. `$WERF_SECONDARY_REPO_1=...`, `$WERF_SECONDARY_REPO_2=...`)
- `--skip-image-spec-stage`
Force skipping "imageSpec" build stage (default `$WERF_SKIP_IMAGE_SPEC_STAGE` or false)
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default `$WERF_SKIP_TLS_VERIFY_REGISTRY`)
- `--synchronization, -S`
Address of synchronizer for multiple werf processes to work with a single repo.

Default:

- `$WERF_SYNCHRONIZATION`, or
- `:local` if `--repo` is not specified, or
- `https://delivery-sync.deckhouse.ru` if `--repo` has been specified.

The same address should be specified for all werf processes that work with a single repo. `:local` address allows execution of werf processes from a single host only

- `--tmp-dir`
Use specified dir to store tmp files and dirs (default `$WERF_TMP_DIR` or system tmp dir)
- `--to`
Destination address to copy stages to. Use `archive:PATH` for stage archive or `[docker://]REPO` for container registry.
- `--use-build-report`
Use build report, previously saved with `--save-build-report` (by default `$WERF_USE_BUILD_REPORT` or false). Its path and format configured with `--build-report-path`
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit synchronization

Usage:

```
d8 delivery-kit synchronization [options]
```

Run synchronization server

Options

- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--dev`
Enable development mode (default `$WERF_DEV`).
The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
Set dev git branch name (default `$WERF_DEV_BRANCH` or `"_werf-dev"`)
- `--dev-ignore`
Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with `$WERF_DEV_IGNORE_*` (e.g. `$WERF_DEV_IGNORE_TESTS=*_test.go`, `$WERF_DEV_IGNORE_DOCS=path/to/docs`)
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/werf`)
- `--host`
Bind synchronization server to the specified host (default `localhost` or `$WERF_HOST`)

- `--kube-config`
Kubernetes config file path (default \$WERF_KUBE_CONFIG, or \$WERF_KUBECONFIG, or \$KUBECONFIG)
- `--kube-config-base64`
Kubernetes config data as base64 string (default \$WERF_KUBE_CONFIG_BASE64 or \$WERF_KUBECONFIG_BASE64 or \$KUBECONFIG_BASE64)
- `--kube-context`
Kubernetes config context (default \$WERF_KUBE_CONTEXT)
- `--kubernetes`
Use kubernetes lock-manager stages-storage-cache (default \$WERF_KUBERNETES)
- `--kubernetes-namespace-prefix`
Use specified prefix for namespaces created for lock-manager and stages-storage-cache (defaults to 'werf-synchronization-' when --kubernetes option is used or \$WERF_KUBERNETES_NAMESPACE_PREFIX)
- `--local`
Use file lock-manager and file stages-storage-cache (true by default or \$WERF_LOCAL)
- `--local-lock-manager-base-dir`
Use specified directory as base for file lock-manager
(~/werf/synchronization_server/lock_manager by default or \$WERF_LOCAL_LOCK_MANAGER_BASE_DIR)
- `--local-stages-storage-cache-base-dir`
Use specified directory as base for file stages-storage-cache
(~/werf/synchronization_server/stages_storage_cache by default or \$WERF_LOCAL_STAGES_STORAGE_CACHE_BASE_DIR)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default \$WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
Enable debug (default \$WERF_LOG_DEBUG).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default \$WERF_LOG_PRETTY or true).
- `--log-quiet`
Disable explanatory output (default \$WERF_LOG_QUIET).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* \$WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default \$WERF_LOG_TIME or false).
- `--log-time-format`
Specify custom log time format (default \$WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
Enable verbose output (default \$WERF_LOG_VERBOSE).
- `--loose-giterminism`
Loose werf giterminism mode restrictions
- `--port`
Bind synchronization server to the specified port (default 55581 or \$WERF_PORT)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)

- `--ttl`
Time to live for lock-manager locks and stages-storage-cache records (default `$WERF_TTL`)
- `--verbose`
Enable verbose output (default `$WERF_VERBOSE`).

d8 delivery-kit verify

Usage:

```
d8 delivery-kit verify --image-ref <image-ref> [--image-ref <image-ref>...] --verify-roots=<root-ca.pem>
[--verify-manifest] [--verify-elf-files] [options]
```

Verify images

Options

- `--container-registry-mirror`
(Buildah-only) Use specified mirrors for docker.io
- `--debug`
Enable debug (default `$WERF_DEBUG`).
- `--docker-config`
Specify docker config directory path. Default `$WERF_DOCKER_CONFIG` or `$DOCKER_CONFIG` or `~/.docker` (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--home-dir`
Use specified dir to store werf cache files and dirs (default `$WERF_HOME` or `~/.werf`)
- `--image-ref`
Verify only passed references (default `$WERF_IMAGE_REF` separated by comma)
- `--insecure-registry`
Use plain HTTP requests when accessing a registry (default `$WERF_INSECURE_REGISTRY`)
- `--log-color-mode`
Set log color mode.
Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
Default `$WERF_LOG_COLOR_MODE` or auto mode.
- `--log-debug`
Enable debug (default `$WERF_LOG_DEBUG`).
- `--log-pretty`
Enable emojis, auto line wrapping and log process border (default `$WERF_LOG_PRETTY` or true).
- `--log-project-dir`
Print current project directory path (default `$WERF_LOG_PROJECT_DIR`)
- `--log-quiet`
Disable explanatory output (default `$WERF_LOG_QUIET`).
- `--log-terminal-width`
Set log terminal width.
Defaults to:
* `$WERF_LOG_TERMINAL_WIDTH`
* interactive terminal width or 140
- `--log-time`
Add time to log entries for precise event time tracking (default `$WERF_LOG_TIME` or false).
- `--log-time-format`
Specify custom log time format (default `$WERF_LOG_TIME_FORMAT` or RFC3339 format).
- `--log-verbose`
Enable verbose output (default `$WERF_LOG_VERBOSE`).
- `--parallel, -p`
Run in parallel (default `$WERF_PARALLEL` or true)

- `--parallel-tasks-limit`
Parallel tasks limit, set -1 to remove the limitation (default \$WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`
Disable explanatory output (default \$WERF_QUIET).
- `--skip-tls-verify-registry`
Skip TLS certificate validation when accessing a registry (default \$WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`
Use specified dir to store tmp files and dirs (default \$WERF_TMP_DIR or system tmp dir)
- `--verbose`
Enable verbose output (default \$WERF_VERBOSE).
- `--verify-bsign-elf-files`
Enable ELF files verification for bsign signed files (default \$WERF_VERIFY_BSIGN_ELF_FILES).
- `--verify-elf-files`
Enable ELF files verification (default \$WERF_VERIFY_ELF_FILES).
When enabled, the root certificates must be specified with --verify-roots option
- `--verify-manifest`
Enable image manifest verification (default \$WERF_VERIFY_MANIFEST).
When enabled,
the root certificates must be specified with --verify-roots option
- `--verify-roots`
The root certificates as path to PEM file or base64-encoded PEM (default \$WERF_VERIFY_ROOTS separated by comma)

d8 delivery-kit version

Usage:

```
d8 delivery-kit version [options]
```

Print version

d8 help

Help about any command

d8 iam

Manage Deckhouse users, groups, and access grants

d8 iam access

Usage:

```
d8 iam access [options]
```

Grant or revoke access (current authz model)

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam access grant

Usage:

```
d8 iam access grant (user|group) NAME --access-level LEVEL [scope] [options]
```

Grant access to a user or group (current authz model)

Options

- `--access-level`
Access level: User, PrivilegedUser, Editor, Admin, ClusterEditor, ClusterAdmin, SuperAdmin
- `--allow-scale`
Allow scaling workloads
- `--dry-run`
Print the resource(s) that would be created without applying
- `--namespace, -n`
Target namespace(s) for namespaced grants (repeatable). Mutually exclusive with `--scope`
- `--output, -o`
Output format: name|yaml|json
- `--port-forwarding`
Allow port-forwarding
- `--scope`
Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with `-n/-namespace`

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam access revoke

Usage:

```
d8 iam access revoke (user|group) NAME --access-level LEVEL [scope] [options]
```

Revoke a d8-managed access grant

Options

- `--access-level`
Access level to revoke
- `--allow-scale`
Match grants with allow-scale enabled
- `--dry-run`
Resolve and print the object(s) that would be deleted without touching the cluster
- `--namespace, -n`
Target namespace(s). Mutually exclusive with `--scope`
- `--output, -o`
Output format: name|yaml|json
- `--port-forwarding`
Match grants with port-forwarding enabled
- `--scope`
Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with `-n/-namespace`

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam get

Usage:

```
d8 iam get (user|group|rule) NAME [options]
```

Show a single Deckhouse IAM object

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam get group

Alias: `groups`

Usage:

```
d8 iam get (user|group|rule) NAME group <name> [options]
```

Show detailed access for a specific group

Options

- `--output, -o`
Output format: table|json|yaml

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam get rule

Alias: `rules`

Usage:

```
d8 iam get (user|group|rule) NAME rule REF [options]
```

Show a single CAR or AR with subjects, scope and reverse CR lookup

Options

- `--output, -o`
Output format: text|json|yaml

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam get user

Alias: `users`

Usage:

```
d8 iam get (user|group|rule) NAME user <name> [options]
```

Show detailed access for a specific user

Options

- `--output, -o`
Output format: table|json|yaml

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam group

Alias: `groups`

Usage:

```
d8 iam group [options]
```

Manage Deckhouse local groups (user-authn)

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam group add-member

Usage:

```
d8 iam group add-member GROUP [user|group] MEMBER [options]
```

Add a member (user or group) to a local group

Options

- `--create-group`
Create the target group if it does not exist

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam group create

Usage:

```
d8 iam group create <name> [options]
```

Create a local group in Deckhouse

Options

- `--dry-run`
Print the resource that would be created without applying
- `--output, -o`
Output format: name|yaml|json

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam group delete**Usage:**

```
d8 iam group delete <name> [options]
```

Delete a local group

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam group remove-member**Usage:**

```
d8 iam group remove-member GROUP [user|group] MEMBER [options]
```

Remove a member (user or group) from a local group

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam list**Usage:**

```
d8 iam list (users|groups|rules) [options]
```

List Deckhouse IAM objects

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam list groups

Alias: `group`

Usage:

```
d8 iam list (users|groups|rules) groups [options]
```

List all groups with their effective access

Options

- `--output, -o`
Output format: table|json

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam list rules**Alias:** `rule`**Usage:**

```
d8 iam list (users|groups|rules) rules [options]
```

List ClusterAuthorizationRules and AuthorizationRules

Options

- `--cluster`
Only ClusterAuthorizationRules
- `--managed-only`
Only rules managed by d8-cli (label app.kubernetes.io/managed-by=d8-cli)
- `--manual-only`
Only rules NOT managed by d8-cli
- `--namespace, -n`
Restrict to AuthorizationRules in these namespaces (repeatable). Implicitly disables CARs unless --cluster is also set.
- `--output, -o`
Output format: table|json|yaml

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam list users**Alias:** `user`**Usage:**

```
d8 iam list (users|groups|rules) users [options]
```

List all users with their effective access

Options

- `--output, -o`
Output format: table|json

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user**Alias:** `userop`

Usage:

```
d8 iam user [options]
```

Manage Deckhouse users (user-authn)

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user create**Usage:**

```
d8 iam user create <name> --email <email> [flags] [options]
```

Create a local static user in Deckhouse

Options

- `--create-groups`
Create groups specified by --member-of if they do not exist
- `--dry-run`
Print the resource that would be created without applying
- `--email`
User email address (required, must be lowercase)
- `--generate-password`
Auto-generate a strong password (shown once on stderr)
- `--member-of`
Add user to these groups (repeatable)
- `--output, -o`
Output format: name|yaml|json
- `--password-hash`
Use a pre-computed bcrypt hash directly (must start with \$2 and be a valid bcrypt cost)
- `--password-prompt`
Read password interactively with hidden input
- `--password-stdin`
Read password from stdin (for CI/pipelines)
- `--ttl`
User time-to-live (e.g. 24h, 30m). Can only be set once; expireAt will not update on change.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user delete**Usage:**

```
d8 iam user delete <name> [options]
```

Delete a local static user

Options

- `--keep-memberships`

Do not remove the user from local Group spec.members; only delete the User CR

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user lock

Usage:

```
d8 iam user lock <username> <lockDuration> [options]
```

Lock local user in Dex for a period of time

Options

- `--timeout`
How long to wait for completion when --wait is enabled.
- `--wait`
Wait for UserOperation completion and print result.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user reset-password

Alias: `resetpass`

Usage:

```
d8 iam user reset-password <username> [options]
```

Reset local user's password in Dex

Options

- `--generate-password`
Auto-generate a strong password (shown once on stderr)
- `--password-hash`
Use a pre-computed bcrypt hash directly (must start with \$2 and be a valid bcrypt cost)
- `--password-prompt`
Read password interactively with hidden input
- `--password-stdin`
Read password from stdin (for CI/pipelines)
- `--timeout`
How long to wait for completion when --wait is enabled.
- `--wait`
Wait for UserOperation completion and print result.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user reset2fa

Usage:

```
d8 iam user reset2fa <username> [options]
```

Reset local user's 2FA (TOTP) in Dex

Options

- `--timeout`
How long to wait for completion when `--wait` is enabled.
- `--wait`
Wait for UserOperation completion and print result.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 iam user unlock

Usage:

```
d8 iam user unlock <username> [options]
```

Unlock local user in Dex

Options

- `--timeout`
How long to wait for completion when `--wait` is enabled.
- `--wait`
Wait for UserOperation completion and print result.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 k

Alias: `kubectl`

kubectl controls the Kubernetes cluster manager

Options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority

- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

Usage:

```
d8 k alpha [options]
```

Commands for features in alpha

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k annotate

Usage:

```
d8 k annotate [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-version=version] [options]
```

Update the annotations on a resource

Options

- `--all`

Select all resources, in the namespace of the specified resource types.

- `--all-namespaces`, `-A`

If true, check the specified action in all namespaces.

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--field-selector`

Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.

- `--filename`, `-f`

Filename, directory, or URL to files identifying the resource to update the annotation

- `--kustomize`, `-k`

Process the kustomization directory. This flag can't be used together with `-f` or `-R`.

- `--list`

If true, display the annotations for a given resource.

- `--local`

If true, annotation will NOT contact api-server but run locally.

- `--output`, `-o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--overwrite`

If true, allow annotations to be overwritten, otherwise reject annotation updates that overwrite existing annotations.

- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`
If non-empty, the annotation update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-resources

Usage:

```
d8 k api-resources [options]
```

Print the supported API resources on the server

Options

- `--api-group`

Limit to resources in the specified API group.

- `--cached`

Use the cached list of resources if available.

- `--categories`

Limit to resources that belong to the specified categories.

- `--namespaced`

If false, non-namespaced resources will be returned, otherwise returning namespaced resources by default.

- `--no-headers`

When using the default or custom-column output format, don't print headers (default print headers).

- `--output, -o`

Output format. One of: (json, yaml, name, wide).

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--sort-by`

If non-empty, sort list of resources using specified field. The field can be either 'name' or 'kind'.

- `--verbs`

Limit to resources that support the specified verbs.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-versions

Usage:

```
d8 k api-versions [options]
```

Print the supported API versions on the server, in the form of "group/version"

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) [options]
```

Apply a configuration to a resource by file name or stdin

Options

- `--all`
Select all resources in the namespace of the specified resource types.
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cascade`
Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

- Name of the manager used to track field ownership.
- `--filename, -f`

The files that contain the configurations to apply.
- `--force`

If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--force-conflicts`

If true, server-side apply will force the changes against conflicts.
- `--grace-period`

Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when `--force` is true (force deletion).
- `--kustomize, -k`

Process a kustomization directory. This flag can't be used together with `-f` or `-R`.
- `--openapi-patch`

If true, use openapi to calculate diff when the openapi presents and the resource can be found in the openapi spec. Otherwise, fall back to use baked-in types.
- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

Automatically resolve conflicts between the modified and live configuration by using values from the modified configuration
- `--prune`

Automatically delete resource objects, that do not appear in the configs and are created by either apply or create `--save-config`. Should be used with either `-l` or `--all`.
- `--prune-allowlist`

Overwrite the default allowlist with `for --prune`
- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`

Process the directory used in `-f, --filename` recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.
- `--server-side`

If true, apply runs in the server instead of the client.
- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

If specified, apply will operate on the subresource of the requested object. Only allowed when using `--server-side`.
- `--template`

Template string or path to template file to use when `-o=go-template, -o=go-template-file`. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--timeout`

The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to

validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--wait`

If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply edit-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) edit-last-applied (RESOURCE/NAME | -f FILENAME) [options]
```

Edit latest last-applied-configuration annotations of a resource/object

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

Filename, directory, or URL to files to use to edit the resource

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to

validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--windows-line-endings`

Defaults to the line ending native to your platform.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply set-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) set-last-applied -f FILENAME [options]
```

Set the last-applied-configuration annotation on a live object to match the contents of a file

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.

- `--create-annotation`

Will create 'last-applied-configuration' annotations if current objects doesn't have one

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--filename, -f`

Filename, directory, or URL to files that contains the last-applied-configuration annotations

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text

log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply view-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) view-last-applied (TYPE [NAME | -l label] | TYPE/NAME | -f FILENAME) [options]
```

View the latest last-applied-configuration annotations of a resource/object

Options

- `--all`
Select all resources in the namespace of the specified resource types
- `--filename, -f`
Filename, directory, or URL to files that contains the last-applied-configuration annotations
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. Must be one of (yaml, json)
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k attach

Usage:

```
d8 k attach (POD | TYPE/NAME) -c CONTAINER [options]
```

Attach to a running container

Options

- `--container, -c`

Container name. If omitted, use the `kubectl.kubernetes.io/default-container` annotation for selecting the container to be attached or the first container in the pod will be chosen

- `--pod-running-timeout`

The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

- `--quiet, -q`

Only print output from the remote session

- `--stdin, -i`

Pass stdin to the container

- `--tty, -t`

Stdin is a TTY

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to

contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth

Usage:

```
d8 k auth [options]
```

Inspect authorization

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth can-i

Usage:

```
d8 k auth can-i VERB [TYPE | TYPE/NAME | NONRESOURCEURL] [options]
```

Check whether an action is allowed

Options

- `--all-namespaces, -A`
If true, check the specified action in all namespaces.
- `--list`
If true, prints all allowed actions.
- `--no-headers`
If true, prints allowed actions without headers
- `--quiet, -q`
If true, suppress output and just return the exit code.
- `--subresource`
SubResource such as pod/log or deployment/scale

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a

- namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server

- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth reconcile

Usage:

```
d8 k auth reconcile -f FILENAME [options]
```

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to reconcile.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--remove-extra-permissions`
If true, removes extra permissions added to roles
- `--remove-extra-subjects`
If true, removes extra subjects added to rolebindings
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority

- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth whoami

Usage:

```
d8 k auth whoami [options]
```

Experimental: Check self subject attributes

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--output`, `-o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace`, `-n`

- If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k autoscale

Usage:

```
d8 k autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu=CPU] [--memory=MEMORY] [options]
```

Auto-scale a deployment, replica set, stateful set, or replication controller

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--cpu`
Target CPU utilization over all the pods. When specified as a percentage (e.g. "70%" for 70% of requested CPU) it will target average utilization. When specified as quantity (e.g. "500m" for 500 milliCPU) it will target average value. Value without units is treated as a quantity with millicPU being the unit (e.g. "500" is "500m").
- `--cpu-percent`
The target average CPU utilization (represented as a percent of requested CPU) over all the pods. If it's not specified or negative, a default autoscaling policy will be used.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.

- `--filename, -f`
Filename, directory, or URL to files identifying the resource to autoscale.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with `-f` or `-R`.
- `--max`
The upper limit for the number of pods that can be set by the autoscaler. Required.
- `--memory`
Target memory utilization over all the pods. When specified as a percentage (e.g. "60%" for 60% of requested memory) it will target average utilization. When specified as quantity (e.g. "200Mi" for 200 MiB, "1Gi" for 1 GiB) it will target average value. Value without units is treated as a quantity with mebibytes being the unit (e.g. "200" is "200Mi").
- `--min`
The lower limit for the number of pods that can be set by the autoscaler. If it's not specified or negative, the server will apply a default value.
- `--name`
The name for the newly created object. If not specified, the name of the input resource will be used.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in `-f`, `--filename` recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS

- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate

Usage:

```
d8 k certificate SUBCOMMAND [options]
```

Modify certificate resources

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`

- The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate approve

Usage:

```
d8 k certificate SUBCOMMAND approve (-f FILENAME | NAME) [options]
```

Approve a certificate signing request

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to update
- `--force`
Update the CSR even if it is already approved.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS

- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate deny

Usage:

```
d8 k certificate SUBCOMMAND deny (-f FILENAME | NAME) [options]
```

Deny a certificate signing request

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to update
- `--force`
Update the CSR even if it is already denied.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.

- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info

Usage:

```
d8 k cluster-info [options]
```

Display cluster information

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory

- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info dump

Usage:

```
d8 k cluster-info dump [options]
```

Dump relevant information for debugging and diagnosis

Options

- `--all-namespaces, -A`
If true, dump all namespaces. If true, `--namespaces` is ignored.
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to `golang` and `jsonpath` output formats.
- `--namespaces`
A comma separated list of namespaces to dump.
- `--output, -o`
Output format. One of: (`json`, `yaml`, `name`, `go-template`, `go-template-file`, `template`, `templatefile`, `jsonpath`, `jsonpath-as-json`, `jsonpath-file`).
- `--output-directory`
Where to output the files. If empty or `'-'` uses stdout, otherwise creates a directory hierarchy in that directory
- `--pod-running-timeout`
The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--show-managed-fields`
If true, keep the `managedFields` when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is `golang` templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k completion

Usage:

```
d8 k completion SHELL [options]
```

Output shell completion code for the specified shell (bash, zsh, fish, or powershell)

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config

Usage:

```
d8 k config SUBCOMMAND [options]
```

Modify kubeconfig files

Options

- `--kubeconfig`
use a particular kubeconfig file

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config current-context

Usage:

```
d8 k config SUBCOMMAND current-context [options]
```

Display the current-context

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-cluster

Usage:

```
d8 k config SUBCOMMAND delete-cluster NAME [options]
```

Delete the specified cluster from the kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-context

Usage:

```
d8 k config SUBCOMMAND delete-context NAME [options]
```

Delete the specified context from the kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-user

Usage:

```
d8 k config SUBCOMMAND delete-user NAME [options]
```

Delete the specified user from the kubeconfig

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

use a particular kubeconfig file

- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-clusters

Usage:

```
d8 k config SUBCOMMAND get-clusters [options]
```

Display clusters defined in the kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory

- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-contexts

Usage:

```
d8 k config SUBCOMMAND get-contexts [(-o|--output=name)] [options]
```

Describe one or many contexts

Options

- `--no-headers`
When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`
Output format. One of: (name).

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace`, `-n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-users

Usage:

```
d8 k config SUBCOMMAND get-users [options]
```

Display users defined in the kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

use a particular kubeconfig file

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config rename-context

Usage:

```
d8 k config SUBCOMMAND rename-context CONTEXT_NAME NEW_NAME [options]
```

Rename a context from the kubeconfig file

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set

Usage:

```
d8 k config SUBCOMMAND set PROPERTY_NAME PROPERTY_VALUE [options]
```

Set an individual value in a kubeconfig file

Options

- `--set-raw-bytes`

When writing a []byte PROPERTY_VALUE, write the given string directly without base64 decoding.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

use a particular kubeconfig file

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-cluster

Usage:

```
d8 k config SUBCOMMAND set-cluster NAME [--server=server] [--certificate-
authority=path/to/certificate/authority] [--insecure-skip-tls-verify=true] [--tls-server-
name=example.com] [options]
```

Set a cluster entry in kubeconfig

Options

- `--certificate-authority`
Path to certificate-authority file for the cluster entry in kubeconfig
- `--embed-certs`
embed-certs for the cluster entry in kubeconfig
- `--insecure-skip-tls-verify`
insecure-skip-tls-verify for the cluster entry in kubeconfig
- `--proxy-url`
proxy-url for the cluster entry in kubeconfig
- `--server`
server for the cluster entry in kubeconfig
- `--tls-server-name`
tls-server-name for the cluster entry in kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-context

Usage:

```
d8 k config SUBCOMMAND set-context [NAME | --current] [--cluster=cluster_nickname] [--user=user_nickname]
[--namespace=namespace] [options]
```

Set a context entry in kubeconfig

Options

- `--cluster`
cluster for the context entry in kubeconfig
- `--current`
Modify the current context
- `--namespace`
namespace for the context entry in kubeconfig
- `--user`
user for the context entry in kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-credentials

Usage:

```
d8 k config SUBCOMMAND set-credentials NAME [--client-certificate=path/to/certfile] [--client-key=path/to/keyfile] [--token=bearer_token] [--username=basic_user] [--password=basic_password] [--auth-provider=provider_name] [--auth-provider-arg=key=value] [--exec-command=exec_command] [--exec-api-version=exec_api_version] [--exec-arg=arg] [--exec-env=key=value] [options]
```

Set a user entry in kubeconfig

Options

- `--auth-provider`

Auth provider for the user entry in kubeconfig

- `--auth-provider-arg`

'key=value' arguments for the auth provider

- `--client-certificate`

Path to client-certificate file for the user entry in kubeconfig

- `--client-key`

Path to client-key file for the user entry in kubeconfig

- `--embed-certs`

Embed client cert/key for the user entry in kubeconfig

- `--exec-api-version`

API version of the exec credential plugin for the user entry in kubeconfig

- `--exec-arg`

New arguments for the exec credential plugin command for the user entry in kubeconfig

- `--exec-command`

Command for the exec credential plugin for the user entry in kubeconfig

- `--exec-env`

'key=value' environment values for the exec credential plugin

- `--exec-interactive-mode`

InteractiveMode of the exec credentials plugin for the user entry in kubeconfig

- `--exec-provide-cluster-info`

ProvideClusterInfo of the exec credentials plugin for the user entry in kubeconfig

- `--password`

password for the user entry in kubeconfig

- `--token`

token for the user entry in kubeconfig

- `--username`

username for the user entry in kubeconfig

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--user`
The name of the kubeconfig user to use
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config unset

Usage:

```
d8 k config SUBCOMMAND unset PROPERTY_NAME [options]
```

Unset an individual value in a kubeconfig file

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server

- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config use-context

Alias: `use`

Usage:

```
d8 k config SUBCOMMAND use-context CONTEXT_NAME [options]
```

Set the current-context in a kubeconfig file

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
use a particular kubeconfig file
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config view

Usage:

```
d8 k config SUBCOMMAND view [options]
```

Display merged kubeconfig settings or a specified kubeconfig file

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--flatten`

Flatten the resulting kubeconfig file into self-contained output (useful for creating portable kubeconfig files)

- `--merge`

Merge the full hierarchy of kubeconfig files

- `--minify`

Remove all information not used by current-context from the output

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--raw`

Display raw byte data and sensitive data

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

use a particular kubeconfig file

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cordon

Usage:

```
d8 k cordon NODE [options]
```

Mark node as unschedulable

Options

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--selector`, `-l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cp

Usage:

```
d8 k cp <file-spec-src> <file-spec-dest> [options]
```

Copy files and directories to and from containers

Options

- `--container, -c`
Container name. If omitted, use the `kubectl.kubernetes.io/default-container` annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--no-preserve`
The copied file/directory's ownership and permissions will not be preserved in the container
- `--retries`
Set number of retries to complete a copy operation from a container. Specify 0 to disable or any negative value for infinite retrying. The default is 0 (no retry).

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create

Usage:

```
d8 k create -f FILENAME [options]
```

Create a resource from a file or from stdin

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--edit`

Edit the API resource before creating

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

Filename, directory, or URL to files to use to create the resource

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--raw`

Raw URI to POST to the server. Uses the transport specified by the kubeconfig file.

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related

manifests organized within the same directory.

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform `kubectl apply` on this object in the future.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'. (e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.

- `--show-managed-fields`

If true, keep the `managedFields` when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is `golang templates` [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: `strict` (or `true`), `warn`, `ignore` (or `false`). `"true"` or `"strict"` will use a schema to validate the input and fail the request if invalid. It will perform server side validation if `ServerSideFieldValidation` is enabled on the api-server, but will fall back to less reliable client-side validation if not. `"warn"` will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as `"ignore"` otherwise. `"false"` or `"ignore"` will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--windows-line-endings`

Only relevant if `--edit=true`. Defaults to the line ending native to your platform.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the `kuberc` file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrole

Usage:

```
d8 k create -f FILENAME clusterrole NAME --verb=verb --resource=resource.group [--resource-name=resourcename] [--dry-run=server|client|none] [options]
```

Create a cluster role

Options

- `--aggregation-rule`

An aggregation label selector for combining ClusterRoles.

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--non-resource-url`
A partial url that user should have access to.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`
Resource that the rule applies to
- `--resource-name`
Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`
Verb that applies to the resources contained in the rule

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS

- connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrolebinding

Usage:

```
d8 k create -f FILENAME clusterrolebinding NAME --clusterrole=NAME [--user=username] [--group=groupname]
[--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a cluster role binding for a particular cluster role

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--clusterrole`
ClusterRole this ClusterRoleBinding should reference
- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--group`

Groups to bind to the clusterrole. The flag can be repeated to add multiple groups.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--serviceaccount`

Service accounts to bind to the clusterrole, in the format `:name`. The flag can be repeated to add multiple service accounts.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--user`

Usernames to bind to the clusterrole. The flag can be repeated to add multiple users.

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if

ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create configmap

Alias: `cm`

Usage:

```
d8 k create -f FILENAME configmap NAME [--from-file=[key]=source] [--from-literal=key1=value1] [--dry-run=server|client|none] [options]
```

Create a config map from a local file, directory or literal value

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--append-hash`
Append a hash of the configmap to its name.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--from-env-file`
Specify the path to a file to read lines of key=val pairs to create a configmap.
- `--from-file`
Key file can be specified using its file path, in which case file basename will be used as configmap key, or optionally with a key and file path, in which case the given key will be used. Specifying a directory will iterate each named file in the directory whose basename is a valid configmap key.
- `--from-literal`
Specify a key and literal value to insert in configmap (i.e. mykey=somevalue)
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create cronjob

Alias: `cj`

Usage:

```
d8 k create -f FILENAME cronjob NAME --image=image --schedule='0/5 * * * ?' -- [COMMAND] [args...]
[options]
```

Create a cron job with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--image`
Image name to run.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--restart`
job's restart policy. supported values: OnFailure, Never
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--schedule`
A schedule in the Cron format the job should be run with.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS

- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create deployment

Alias: `deploy`

Usage:

```
d8 k create -f FILENAME deployment NAME --image=image -- [COMMAND] [args...] [options]
```

Create a deployment with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--image`
Image names to run. A deployment can have multiple images set for multi-container pod.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--port`
The containerPort that this deployment exposes.
- `--replicas, -r`
Number of replicas to create. Default is 1.
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS

- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create ingress

Alias: `ing`

Usage:

```
d8 k create -f FILENAME ingress NAME --rule=host/path=service:port[,tls[=secret]] [options]
```

Create an ingress with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--annotation`
Annotation to insert in the ingress object, in the format annotation=value
- `--class`
Ingress Class to be used
- `--default-backend`
Default service for backend, in format of svcname:port
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--rule`
Rule in format host/path=service:port[,tls=secretname]. Paths containing the leading character '*' are considered pathType=Prefix. tls argument is optional.
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`

- Path to a client certificate file for TLS
- `--client-key`
- Path to a client key file for TLS
- `--cluster`
- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create job

Usage:

```
d8 k create -f FILENAME job NAME --image=image [--from=cronjob/name] -- [COMMAND] [args...] [options]
```

Create a job with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--from`
The name of the resource to create a Job from (only cronjob is supported).
- `--image`
Image name to run.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`

- Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create namespace

Alias: `ns`

Usage:

```
d8 k create -f FILENAME namespace NAME [--dry-run=server|client|none] [options]
```

Create a namespace with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`

- The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create poddisruptionbudget

Alias: `pdb`

Usage:

```
d8 k create -f FILENAME poddisruptionbudget NAME --selector=SELECTOR --min-available=N [--dry-run=server|client|none] [options]
```

Create a pod disruption budget with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to go lang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--max-unavailable`
The maximum number or percentage of unavailable pods this budget requires.
- `--min-available`
The minimum number or percentage of available pods this budget requires.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`
A label selector to use for this budget. Only equality-based selector requirements are supported.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create priorityclass

Alias: `pc`

Usage:

```
d8 k create -f FILENAME priorityclass NAME --value=VALUE --global-default=B00L [--dry-run=server|client|none] [options]
```

Create a priority class with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--description`
description is an arbitrary string that usually provides guidelines on when this priority class should be used.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--global-default`
global-default specifies whether this PriorityClass should be considered as the default priority.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--preemption-policy`
preemption-policy is the policy for preempting pods with lower priority.
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--value`
the value of this priority class.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`

- Path to a client certificate file for TLS
- `--client-key`
- Path to a client key file for TLS
- `--cluster`
- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create quota

Alias: resourcequota

Usage:

```
d8 k create -f FILENAME quota NAME [--hard=key1=value1,key2=value2] [--scopes=Scope1,Scope2] [--dry-run=server|client|none] [options]
```

Create a quota with the specified name

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--hard`

A comma-delimited set of resource=quantity pairs that define a hard limit.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--scopes`

A comma-delimited set of quota scopes that must all match each object tracked by the quota.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if

ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create role

Usage:

```
d8 k create -f FILENAME role NAME --verb=verb --resource=resource.group/subresource [--resource-name=resourcename] [--dry-run=server|client|none] [options]
```

Create a role with single rule

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`
Resource that the rule applies to
- `--resource-name`
Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`
Verb that applies to the resources contained in the rule

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`

- Path to a client certificate file for TLS
- `--client-key`
- Path to a client key file for TLS
- `--cluster`
- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create rolebinding

Usage:

```
d8 k create -f FILENAME rolebinding NAME --clusterrole=NAME|--role=NAME [--user=username] [--group=groupname] [--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a role binding for a particular role or cluster role

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--clusterrole`
ClusterRole this RoleBinding should reference
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--group`
Groups to bind to the role. The flag can be repeated to add multiple groups.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--role`
Role this RoleBinding should reference
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--serviceaccount`
Service accounts to bind to the role, in the format namespace:serviceaccountname. The flag can be repeated to add multiple service accounts.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`
Usernames to bind to the role. The flag can be repeated to add multiple users.
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) [options]
```

Create a secret using a specified subcommand

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server

- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret docker-registry

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) docker-registry NAME --docker-username=user --docker-password=password --docker-email=email [--docker-server=string] [--from-file=[key=]source] [--dry-run=server|client|none] [options]
```

Create a secret for use with a Docker registry

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--append-hash`
Append a hash of the secret to its name.
- `--docker-email`
Email for Docker registry
- `--docker-password`
Password for Docker registry authentication
- `--docker-server`
Server location for Docker registry
- `--docker-username`
Username for Docker registry authentication
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--from-file`
Key files can be specified using their file path, in which case a default name of .dockerconfigjson will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key. For this command, the key should always be .dockerconfigjson.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the

annotation will be unchanged. This flag is useful when you want to perform `kubectl apply` on this object in the future.

- `--show-managed-fields`

If true, keep the `managedFields` when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is `golang templates` [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: `strict` (or `true`), `warn`, `ignore` (or `false`). `"true"` or `"strict"` will use a schema to validate the input and fail the request if invalid. It will perform server side validation if `ServerSideFieldValidation` is enabled on the api-server, but will fall back to less reliable client-side validation if not. `"warn"` will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as `"ignore"` otherwise. `"false"` or `"ignore"` will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

- Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret generic

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) generic NAME [--type=string] [--from-file=[key=]source] [--from-literal=key1=value1] [--dry-run=server|client|none] [options]
```

Create a secret from a local file, directory, or literal value

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--append-hash`
Append a hash of the secret to its name.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--from-env-file`
Specify the path to a file to read lines of key=val pairs to create a secret.
- `--from-file`
Key files can be specified using their file path, in which case a default name will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key.
- `--from-literal`
Specify a key and literal value to insert in secret (i.e. mykey=somevalue)

- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--type`
The type of secret to create
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret tls

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) tls NAME --cert=path/to/cert/file --key=path/to/key/file [--dry-run=server|client|none] [options]
```

Create a TLS secret

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--append-hash`
Append a hash of the secret to its name.
- `--cert`
Path to PEM encoded public key certificate.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--key`

Path to private key associated with given certificate.

- `--output`, `-o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform `kubectl apply` on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is `golang templates` [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: `strict` (or `true`), `warn`, `ignore` (or `false`). `"true"` or `"strict"` will use a schema to validate the input and fail the request if invalid. It will perform server side validation if `ServerSideFieldValidation` is enabled on the api-server, but will fall back to less reliable client-side validation if not. `"warn"` will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as `"ignore"` otherwise. `"false"` or `"ignore"` will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service

Alias: `svc`

Usage:

```
d8 k create -f FILENAME service [options]
```

Create a service using a specified subcommand

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

- Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service clusterip

Usage:

```
d8 k create -f FILENAME service clusterip NAME [--tcp=<port>:<targetPort>] [--dry-run=server|client|none]
[options]
```

Create a ClusterIP service

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--clusterip`
Assign your own ClusterIP or set to 'None' for a 'headless' service (no loadbalancing).
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`
Port pairs can be specified as ':'.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority

- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service externalname

Usage:

```
d8 k create -f FILENAME service externalname NAME --external-name external.name [--dry-run=server|client|none] [options]
```

Create an ExternalName service

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--external-name`

External name of service

- `--field-manager`

Name of the manager used to track field ownership.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--tcp`

Port pairs can be specified as ':'.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if

ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service loadbalancer

Usage:

```
d8 k create -f FILENAME service loadbalancer NAME [--tcp=port:targetPort] [--dry-run=server|client|none]
[options]
```

Create a LoadBalancer service

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--tcp`

Port pairs can be specified as ':'.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates (<http://golang.org/pkg/text/template/#pkg-overview>).

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service nodeport

Usage:

```
d8 k create -f FILENAME service nodeport NAME [--tcp=port:targetPort] [--dry-run=server|client|none]
[options]
```

Create a NodePort service

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--node-port`
Port used to expose the service on each node in a cluster.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`
Port pairs can be specified as ':'.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`

- The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create serviceaccount

Alias: `sa`

Usage:

```
d8 k create -f FILENAME serviceaccount NAME [--dry-run=server|client|none] [options]
```

Create a service account with the specified name

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`
Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS

- connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create token

Usage:

```
d8 k create -f FILENAME token SERVICE_ACCOUNT_NAME [options]
```

Request a service account token

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--audience`
Audience of the requested token. If unset, defaults to requesting a token for use with the Kubernetes API server. May be repeated to request a token valid for multiple audiences.

- `--bound-object-kind`
Kind of an object to bind the token to. Supported kinds are Node, Pod, Secret. If set, `--bound-object-name` must be provided.
- `--bound-object-name`
Name of an object to bind the token to. The token will expire when the object is deleted. Requires `--bound-object-kind`.
- `--bound-object-uid`
UID of an object to bind the token to. Requires `--bound-object-kind` and `--bound-object-name`. If unset, the UID of the existing object is used.
- `--duration`
Requested lifetime of the issued token. If not set or if set to 0, the lifetime will be determined by the server automatically. The server may return a token with a longer or shorter lifetime.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes

- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k debug

Usage:

```
d8 k debug (POD | TYPE[.VERSION].GROUP)/NAME [ -- COMMAND [args...] ] [options]
```

Create debugging sessions for troubleshooting workloads and nodes

Options

- `--arguments-only`
If specified, everything after -- will be passed to the new container as Args instead of Command.
- `--attach`
If true, wait for the container to start running, and then attach as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true.
- `--container, -c`
Container name to use for debug container.
- `--copy-to`
Create a copy of the target Pod with this name.
- `--custom`
Path to a JSON or YAML file containing a partial container spec to customize built-in debug profiles.
- `--env`
Environment variables to set in the container.
- `--filename, -f`

identifying the resource to debug

- `--image`

Container image to use for debug container. If not specified, the platform's recommended image will be used.

- `--image-pull-policy`

The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.

- `--keep-annotations`

If true, keep the original pod annotations.(This flag only works when used with '--copy-to')

- `--keep-init-containers`

Run the init containers for the pod. Defaults to true.(This flag only works when used with '--copy-to')

- `--keep-labels`

If true, keep the original pod labels.(This flag only works when used with '--copy-to')

- `--keep-liveness`

If true, keep the original pod liveness probes.(This flag only works when used with '--copy-to')

- `--keep-readiness`

If true, keep the original pod readiness probes.(This flag only works when used with '--copy-to')

- `--keep-startup`

If true, keep the original startup probes.(This flag only works when used with '--copy-to')

- `--profile`

Options are "legacy", "general", "baseline", "netadmin", "restricted" or "sysadmin".

- `--quiet, -q`

If true, suppress informational messages.

- `--replace`

When used with '--copy-to', delete the original Pod.

- `--same-node`

When used with '--copy-to', schedule the copy of target Pod on the same node.

- `--set-image`

When used with '--copy-to', a list of name=image pairs for changing container images, similar to how 'kubectrl set image' works.

- `--share-processes`

When used with '--copy-to', enable process namespace sharing in the copy.

- `--stdin, -i`

Keep stdin open on the container(s) in the pod, even if nothing is attached.

- `--target`

When using an ephemeral container, target processes in this container name.

- `--tty, -t`

Allocate a TTY for the debugging container.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

- Path to a client certificate file for TLS
- `--client-key`
- Path to a client key file for TLS
- `--cluster`
- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k delete

Usage:

```
d8 k delete ([-f FILENAME] | [-k DIRECTORY] | TYPE [(NAME | -l label | --all)]) [options]
```

Delete resources by file names, stdin, resources and names, or by resources and label selector

Options

- `--all`
Delete all resources, in the namespace of the specified resource types.
- `--all-namespaces, -A`
If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with `--namespace`.
- `--cascade`
Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-selector`
Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.
- `--filename, -f`
containing the resource to delete.
- `--force`
If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`
Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when `--force` is true (force deletion).
- `--ignore-not-found`
Treat "resource not found" as a successful delete. Defaults to "true" when `--all` is specified.
- `--interactive, -i`
If true, delete resource only when user confirms.
- `--kustomize, -k`
Process a kustomization directory. This flag can't be used together with `-f` or `-R`.
- `--now`
If true, resources are signaled for immediate shutdown (same as `--grace-period=1`).
- `--output, -o`
Output mode. Use "-o name" for shorter output (resource/name).
- `--raw`
Raw URI to DELETE to the server. Uses the transport specified by the kubeconfig file.
- `--recursive, -R`
Process the directory used in `-f`, `--filename` recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.
- `--timeout`
The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--wait`
If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k describe

Usage:

```
d8 k describe (-f FILENAME | TYPE [NAME_PREFIX | -l label] | TYPE/NAME) [options]
```

Show details of a specific resource or group of resources

Options

- `--all-namespaces, -A`

If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with `--namespace`.

- `--chunk-size`

Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.

- `--filename, -f`

Filename, directory, or URL to files containing the resource to describe

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with `-f` or `-R`.

- `--recursive, -R`

Process the directory used in `-f`, `--filename` recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.

- `--show-events`

If true, display events related to the described object.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

- The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k diff

Usage:

```
d8 k diff -f FILENAME [options]
```

Diff the live version against a would-be applied version

Options

- `--concurrency`

Number of objects to process in parallel when diffing against the live version. Larger number = faster, but more memory, I/O and CPU over that shorter period of time.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

Filename, directory, or URL to files contains the configuration to diff

- `--force-conflicts`

If true, server-side apply will force the changes against conflicts.

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--prune`

Include resources that would be deleted by pruning. Can be used with -l and default shows all resources would be pruned

- `--prune-allowlist`

Overwrite the default allowlist with for --prune

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--server-side`

If true, apply runs in the server instead of the client.

- `--show-managed-fields`

If true, include managed fields in the diff.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k drain

Usage:

```
d8 k drain NODE [options]
```

Drain node in preparation for maintenance

Options

- `--chunk-size`

Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.

- `--delete-emptydir-data`

Continue even if there are pods using emptyDir (local data that will be deleted when the node is drained).

- `--disable-eviction`

Force drain to use delete, even if eviction is supported. This will bypass checking

PodDisruptionBudgets, use with caution.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--force`

Continue even if there are pods that do not declare a controller.

- `--grace-period`

Period of time in seconds given to each pod to terminate gracefully. If negative, the default value specified in the pod will be used.

- `--ignore-daemonsets`

Ignore DaemonSet-managed pods.

- `--pod-selector`

Label selector to filter pods on the node

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--skip-wait-for-delete-timeout`

If pod DeletionTimestamp older than N seconds, skip waiting for the pod. Seconds must be greater than 0 to skip.

- `--timeout`

The length of time to wait before giving up, zero means infinite

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

- Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k edit

Usage:

```
d8 k edit (RESOURCE/NAME | -f FILENAME) [options]
```

Edit a resource on the server

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to go lang and jsonpath output formats.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files to use to edit the resource
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--output-patch`

Output the patch if the resource is edited.

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in `-f, --filename` recursively. Useful when you want to manage related manifests organized within the same directory.

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--subresource`

If specified, edit will operate on the subresource of the requested object.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--windows-line-endings`

Defaults to the line ending native to your platform.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k events

Usage:

```
d8 k events [(-o|--output=)json|yaml|name|go-template|go-template-
file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file] [--for TYPE/NAME] [--watch] [--
types=Normal,Warning] [options]
```

List events

Options

- `--all-namespaces, -A`

If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--chunk-size`

Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.

- `--for`

Filter events to only those pertaining to the specified resource.

- `--no-headers`

When using the default output format, don't print headers.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--types`

Output only events of given types.

- `--watch, -w`

After listing the requested events, watch for more events.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k exec

Usage:

```
d8 k exec (POD | TYPE/NAME) [-c CONTAINER] [flags] -- COMMAND [args...] [options]
```

Execute a command in a container

Options

- `--container, -c`

Container name. If omitted, use the `kubectrl.kubernetes.io/default-container` annotation for selecting the container to be attached or the first container in the pod will be chosen

- `--filename, -f`

to use to exec into the resource

- `--pod-running-timeout`

The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

- `--quiet, -q`

Only print output from the remote session

- `--stdin, -i`

Pass stdin to the container

- `--tty, -t`

Stdin is a TTY

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k explain

Usage:

```
d8 k explain TYPE [--recursive=FALSE|TRUE] [--api-version=api-version-group] [-o|--output=plaintext|plaintext-openapi2] [options]
```

Get documentation for a resource

Options

- `--api-version`

Get different explanations for particular API version (API group/version)

- `--output, -o`

Format in which to render the schema (plaintext, plaintext-openapi2)

- `--recursive`

Print the fields of fields (Currently only 1 level deep)

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k expose

Usage:

```
d8 k expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [options]
```

Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--cluster-ip`
ClusterIP to be assigned to the service. Leave empty to auto-allocate, or set to 'None' to create a headless service.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent,

without sending it. If server strategy, submit server-side request without persisting the resource.

- `--external-ip`

Additional external IP address (not managed by Kubernetes) to accept for the service. If this IP is routed to a node, the service can be accessed by this IP in addition to its generated service IP.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

Filename, directory, or URL to files identifying the resource to expose a service

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--labels, -l`

Labels to apply to the service created by this call.

- `--load-balancer-ip`

IP to assign to the LoadBalancer. If empty, an ephemeral IP will be created and used (cloud-provider specific).

- `--name`

The name for the newly created object.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--override-type`

The method used to override the generated object: json, merge, or strategic.

- `--overrides`

An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.

- `--port`

The port that the service should serve on. Copied from the resource being exposed, if unspecified

- `--protocol`

The network protocol for the service to be created. Default is 'TCP'.

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--selector`

A label selector to use for this service. Only equality-based selector requirements are supported. If empty (the default) infer the selector from the replication controller or replica set.)

- `--session-affinity`

If non-empty, set the session affinity for the service to this; legal values: 'None', 'ClientIP'

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--target-port`

Name or number for the port on the container that the service should direct traffic to. Optional.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--type`

Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k get

Usage:

```
d8 k get [(-o|--output=json|yaml|name|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file|custom-columns|custom-columns-file|wide) (TYPE[.VERSION][.GROUP] [NAME | -l label] | TYPE[.VERSION][.GROUP]/NAME ...)] [flags] [options]
```

Display one or many resources

Options

- `--all-namespaces, -A`

If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with `--namespace`.

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to go lang and jsonpath output formats.

- `--chunk-size`

Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.

- `--field-selector`

Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.

- `--filename, -f`

Filename, directory, or URL to files identifying the resource to get from a server.

- `--ignore-not-found`

If set to true, suppresses NotFound error for specific objects that do not exist. Using this flag with commands that query for collections of resources has no effect when no resources are found.

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with `-f` or `-R`.

- `--label-columns, -L`

Accepts a comma separated list of labels that are going to be presented as columns. Names are case-sensitive. You can also use multiple flag options like `-L label1 -L label2...`

- `--no-headers`

When using the default or custom-column output format, don't print headers (default print headers).

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file, custom-columns, custom-columns-file, wide). See custom columns [<https://kubernetes.io/docs/reference/kubectl/#custom-columns>], go lang template [<http://golang.org/pkg/text/template/#pkg-overview>] and jsonpath template [<https://kubernetes.io/docs/reference/kubectl/jsonpath/>].

- `--output-watch-events`

Output watch event objects when --watch or --watch-only is used. Existing objects are output as initial ADDED events.

- `--raw`

Raw URI to request from the server. Uses the transport specified by the kubeconfig file.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'. (e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--server-print`

If true, have the server return the appropriate table output. Supports extension APIs and CRDs.

- `--show-kind`

If present, list the resource type for the requested object(s).

- `--show-labels`

When printing, show all labels as the last column (default hide labels column)

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--sort-by`

If non-empty, sort list types using this field specification. The field specification is expressed as a JSONPath expression (e.g. '{.metadata.name}'). The field in the API resource specified by this JSONPath expression must be an integer or a string.

- `--subresource`

If specified, gets the subresource of the requested object.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--watch, -w`

After listing/getting the requested object, watch for changes.

- `--watch-only`

Watch for changes to the requested object(s), without listing/getting first.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k kustomize

Usage:

```
d8 k kustomize DIR [options]
```

Build a kustomization target from a directory or URL

Options

- `--as-current-user`
use the uid and gid of the command executor to run the function in the container
- `--enable-alpha-plugins`

- enable kustomize plugins
- `--enable-helm`
Enable use of the Helm chart inflator generator.
- `--enable-managedby-label`
enable adding app.kubernetes.io/managed-by
- `--env, -e`
a list of environment variables to be used by functions
- `--helm-api-versions`
Kubernetes api versions used by Helm for Capabilities.APIVersions
- `--helm-command`
helm command (path to executable)
- `--helm-debug`
Enable debug output from the Helm chart inflator generator.
- `--helm-kube-version`
Kubernetes version used by Helm for Capabilities.KubeVersion
- `--load-restrictor`
if set to 'LoadRestrictionsNone', local kustomizations may load files from outside their root. This does, however, break the relocatability of the kustomization.
- `--mount`
a list of storage options read from the filesystem
- `--network`
enable network access for functions that declare it
- `--network-name`
the docker network to run the container in
- `--output, -o`
If specified, write output to this path.
- `--reorder`
Reorder the resources just before output. Use 'legacy' to apply a legacy reordering (Namespaces first, Webhooks last, etc). Use 'none' to suppress a final reordering.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k label

Usage:

```
d8 k label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-version=version] [options]
```

Update the labels on a resource

Options

- `--all`

Select all resources, in the namespace of the specified resource types

- `--all-namespaces, -A`

If true, check the specified action in all namespaces.

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--field-selector`
Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to update the labels
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`
If true, display the labels for a given resource.
- `--local`
If true, label will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`
If true, allow labels to be overwritten, otherwise reject label updates that overwrite existing labels.
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`
If non-empty, the labels update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`

- Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k logs

Usage:

```
d8 k logs [-f] [-p] (POD | TYPE/NAME) [-c CONTAINER] [options]
```

Print the logs for a container in a pod

Options

- `--all-containers`
Get all containers' logs in the pod(s).
- `--all-pods`
Get logs from all pod(s). Sets prefix to true.
- `--container, -c`
Print the logs of this container
- `--follow, -f`
Specify if the logs should be streamed.
- `--ignore-errors`
If watching / following pod logs, allow for any errors that occur to be non-fatal
- `--insecure-skip-tls-verify-backend`
Skip verifying the identity of the kubelet that logs are requested from. In theory, an attacker could provide invalid log content back. You might want to use this if your kubelet serving certificates have expired.
- `--limit-bytes`
Maximum bytes of logs to return. Defaults to no limit.
- `--max-log-requests`
Specify maximum number of concurrent logs to follow when using by a selector. Defaults to 5.
- `--pod-running-timeout`
The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--prefix`
Prefix each log line with the log source (pod name and container name)
- `--previous, -p`
If true, print the logs for the previous instance of the container in a pod if it exists.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--since`
Only return logs newer than a relative duration like 5s, 2m, or 3h. Defaults to all logs. Only one of since-time / since may be used.
- `--since-time`
Only return logs after a specific date (RFC3339). Defaults to all logs. Only one of since-time / since may be used.
- `--tail`
Lines of recent log file to display. Defaults to -1 with no selector, showing all log lines otherwise 10, if a selector is provided.
- `--timestamps`
Include timestamps on each line in the log output

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k options

Usage:

```
d8 k options [options]
```

Print the list of flags inherited by all commands

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k patch

Usage:

```
d8 k patch (-f FILENAME | TYPE NAME) [-p PATCH|--patch-file FILE] [options]
```

Update fields of a resource

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to update
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`
If true, patch will operate on the content of the file, not the server-side resource.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--patch, -p`
The patch to be applied to the resource JSON file.
- `--patch-file`
A file containing a patch to be applied to the resource.
- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in `-f`, `--filename` recursively. Useful when you want to manage related manifests organized within the same directory.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--subresource`

If specified, patch will operate on the subresource of the requested object.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--type`

The type of patch being provided; one of [json merge strategic]

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin

Usage:

```
d8 k plugin [flags] [options]
```

Provides utilities for interacting with plugins

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin list

Usage:

```
d8 k plugin [flags] list [options]
```

List all visible plugin executables on a user's PATH

Options

- `--name-only`
If true, display only the binary name of each plugin, rather than its full path

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`

- The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k port-forward

Usage:

```
d8 k port-forward TYPE/NAME [options] [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N]
```

Forward one or more local ports to a pod

Options

- `--address`
Addresses to listen on (comma separated). Only accepts IP addresses or localhost as a value. When localhost is supplied, kubectl will try to bind on both 127.0.0.1 and ::1 and will fail if neither of these addresses are available to bind.
- `--pod-running-timeout`
The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k proxy

Usage:

```
d8 k proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [options]
```

Run a proxy to the Kubernetes API server

Options

- `--accept-hosts`

Regular expression for hosts that the proxy should accept.

- `--accept-paths`

Regular expression for paths that the proxy should accept.

- `--address`

The IP address on which to serve on.

- `--api-prefix`

Prefix to serve the proxied API under.

- `--append-server-path`

If true, enables automatic path appending of the kube context server path to each request.

- `--disable-filter`
If true, disable request filtering in the proxy. This is dangerous, and can leave you vulnerable to XSRF attacks, when used with an accessible port.
- `--keepalive`
keepalive specifies the keep-alive period for an active network connection. Set to 0 to disable keepalive.
- `--port`, `-p`
The port on which to run the proxy. Set to 0 to pick a random port.
- `--reject-methods`
Regular expression for HTTP methods that the proxy should reject (example `--reject-methods='POST,PUT,PATCH'`).
- `--reject-paths`
Regular expression for paths that the proxy should reject. Paths specified here will be rejected even accepted by `--accept-paths`.
- `--unix-socket`, `-u`
Unix socket on which to run the proxy.
- `--www`, `-w`
Also serve static files from the given directory under the specified prefix.
- `--www-prefix`, `-P`
Prefix to serve static files under, if static file directory is specified.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes

- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k replace

Usage:

```
d8 k replace -f FILENAME [options]
```

Replace a resource by file name or stdin

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--cascade`
Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
The files that contain the configurations to replace.
- `--force`

If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.

- `--grace-period`

Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when `--force` is true (force deletion).

- `--kustomize, -k`

Process a kustomization directory. This flag can't be used together with `-f` or `-R`.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--raw`

Raw URI to PUT to the server. Uses the transport specified by the kubeconfig file.

- `--recursive, -R`

Process the directory used in `-f`, `--filename` recursively. Useful when you want to manage related manifests organized within the same directory.

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform `kubectl apply` on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--subresource`

If specified, replace will operate on the subresource of the requested object.

- `--template`

Template string or path to template file to use when `-o=go-template`, `-o=go-template-file`. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--timeout`

The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if `ServerSideFieldValidation` is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--wait`

If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

- Path to a client certificate file for TLS
- `--client-key`
- Path to a client key file for TLS
- `--cluster`
- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout

Usage:

```
d8 k rollout SUBCOMMAND [options]
```

Manage the rollout of a resource

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to

contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout history

Usage:

```
d8 k rollout SUBCOMMAND history (TYPE NAME | TYPE/NAME) [flags] [options]
```

View rollout history

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.

- `--filename, -f`

Filename, directory, or URL to files identifying the resource to get from a server.

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--revision`

See the details, including podTemplate of the revision specified

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout pause

Usage:

```
d8 k rollout SUBCOMMAND pause RESOURCE [options]
```

Mark the provided resource as paused

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

Filename, directory, or URL to files identifying the resource to get from a server.

- `--kustomize, -k`

Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout restart

Usage:

```
d8 k rollout SUBCOMMAND restart RESOURCE [options]
```

Restart a resource

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goolang and jsonpath output formats.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goolang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.

- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout resume

Usage:

```
d8 k rollout SUBCOMMAND resume RESOURCE [options]
```

Resume a paused resource

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`

- Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout status

Usage:

```
d8 k rollout SUBCOMMAND status (TYPE NAME | TYPE/NAME) [flags] [options]
```

Show the status of the rollout

Options

- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--revision`
Pin to a specific revision for showing its status. Defaults to 0 (last revision).
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--timeout`
The length of time to wait before ending watch, zero means never. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).
- `--watch, -w`
Watch the status of the rollout until it's done.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use

- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout undo

Usage:

```
d8 k rollout SUBCOMMAND undo (TYPE NAME | TYPE/NAME) [flags] [options]
```

Undo a previous rollout

Options

- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--to-revision`
The revision to rollback to. Default to 0 (last revision).

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k run

Usage:

```
d8 k run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server|client] [--overrides=inline-json] [--command] -- [COMMAND] [args...] [options]
```

Run a particular image on the cluster

Options

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--annotations`

Annotations to apply to the pod.

- `--attach`

If true, wait for the Pod to start running, and then attach to the Pod as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true. With '--restart=Never' the exit code of the container process is returned.

- `--cascade`

Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.

- `--command`

If true and extra arguments are present, use them as the 'command' field in the container, rather than the 'args' field which is the default.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--env`

Environment variables to set in the container.

- `--expose`

If true, create a ClusterIP service associated with the pod. Requires '--port'.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

to use to replace the resource.

- `--force`

If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.

- `--grace-period`

Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).

- `--image`

The image for the container to run.

- `--image-pull-policy`

The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.

- `--kustomize, -k`

Process a kustomization directory. This flag can't be used together with -f or -R.

- `--labels, -l`

Comma separated labels to apply to the pod. Will override previous values.

- `--leave-stdin-open`

If the pod is started in interactive mode or with stdin, leave stdin open after the first attach completes. By default, stdin will be closed after the first attach completes.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--override-type`

The method used to override the generated object: json, merge, or strategic.

- `--overrides`

An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.

- `--pod-running-timeout`

The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

- `--port`

The port that this container exposes.

- `--privileged`

If true, run the container in privileged mode.

- `--quiet, -q`

If true, suppress prompt messages.

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--restart`

The restart policy for this Pod. Legal values [Always, OnFailure, Never].

- `--rm`

If true, delete the pod after it exits. Only valid when attaching to the container, e.g. with '--attach' or with '-i/--stdin'.

- `--save-config`

If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--stdin, -i`

Keep stdin open on the container in the pod, even if nothing is attached.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is go lang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--timeout`

The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object

- `--tty, -t`

Allocate a TTY for the container in the pod.

- `--wait`

If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

- Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
 - `--cache-dir`
Default cache directory
 - `--certificate-authority`
Path to a cert file for the certificate authority
 - `--client-certificate`
Path to a client certificate file for TLS
 - `--client-key`
Path to a client key file for TLS
 - `--cluster`
The name of the kubeconfig cluster to use
 - `--context`
The name of the kubeconfig context to use
 - `--disable-compression`
If true, opt-out of response compression for all requests to the server
 - `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
 - `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
 - `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
 - `--log-flush-frequency`
Maximum number of seconds between log flushes
 - `--match-server-version`
Require server version to match client version
 - `--namespace, -n`
If present, the namespace scope for this CLI request
 - `--password`
Password for basic authentication to the API server
 - `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
 - `--profile-output`
Name of the file to write the profile to
 - `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
 - `--server, -s`
The address and port of the Kubernetes API server
 - `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
 - `--token`
Bearer token for authentication to the API server
 - `--user`
The name of the kubeconfig user to use
 - `--username`
Username for basic authentication to the API server
 - `--v, -v`
number for the log level verbosity

- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k scale

Usage:

```
d8 k scale [--resource-version=version] [--current-replicas=count] --replicas=COUNT (-f FILENAME | TYPE NAME) [options]
```

Set a new size for a deployment, replica set, or replication controller

Options

- `--all`
Select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--current-replicas`
Precondition for current size. Requires that the current size of the resource match this value in order to scale. -1 (default) for no condition.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to set a new size
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--replicas`
The new desired number of replicas. Required.
- `--resource-version`
Precondition for resource version. Requires that the current resource version match this value in order to scale.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].

- `--timeout`

The length of time to wait before giving up on a scale operation, zero means don't wait. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--match-server-version`

Require server version to match client version

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--profile`

Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

Name of the file to write the profile to

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to

- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set

Usage:

```
d8 k set SUBCOMMAND [options]
```

Set specific features on objects

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes

- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set env

Usage:

```
d8 k set SUBCOMMAND env RESOURCE/NAME KEY_1=VAL_1 ... KEY_N=VAL_N [options]
```

Update environment variables on a pod template

Options

- `--all`
If true, select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--containers, -c`
The names of containers in the selected pod templates to change - may use wildcards
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--env, -e`
Specify a key-value pair for an environment variable to set into each container.
- `--field-manager`
Name of the manager used to track field ownership.

- `--filename, -f`
Filename, directory, or URL to files the resource to update the env
- `--from`
The name of a resource from which to inject environment variables
- `--keys`
Comma-separated list of keys to import from specified resource
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`
If true, display the environment and any changes in the standard format. this flag will removed when we have kubectl view env.
- `--local`
If true, set env will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`
If true, allow environment to be overwritten, otherwise reject updates that overwrite existing environment.
- `--prefix`
Prefix to append to variable names
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resolve`
If true, show secret or configmap references when listing variables
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
- Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--match-server-version`
- Require server version to match client version
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--profile`
- Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
- Name of the file to write the profile to
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
- Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set image

Usage:

```
d8 k set SUBCOMMAND image (-f FILENAME | TYPE NAME) CONTAINER_NAME_1=CONTAINER_IMAGE_1 ...
CONTAINER_NAME_N=CONTAINER_IMAGE_N [options]
```

Update the image of a pod template

Options

- `--all`
Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`
If true, set image will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS

- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set resources

Usage:

```
d8 k set SUBCOMMAND resources (-f FILENAME | TYPE NAME) ([--limits=LIMITS & --requests=REQUESTS]
[options])
```

Update resource requests/limits on objects with pod templates

Options

- `--all`
Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.
- `--containers, -c`
The names of containers in the selected pod templates to change, all containers are selected by default - may use wildcards
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--limits`
The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--local`
If true, set resources will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--requests`
The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set selector

Usage:

```
d8 k set SUBCOMMAND selector (-f FILENAME | TYPE NAME) EXPRESSIONS [--resource-version=version] [options]
```

Set the selector on a resource

Options

- `--all`

Select all resources in the namespace of the specified resource types

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--filename, -f`

identifying the resource.

- `--local`

If true, annotation will NOT contact api-server but run locally.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive, -R`

Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.

- `--resource-version`

If non-empty, the selectors update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set serviceaccount

Alias: `sa`

Usage:

```
d8 k set SUBCOMMAND serviceaccount (-f FILENAME | TYPE NAME) SERVICE_ACCOUNT [options]
```

Update the service account of a resource

Options

- `--all`
Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`
If true, set serviceaccount will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`
Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.

- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set subject

Usage:

```
d8 k set SUBCOMMAND subject (-f FILENAME | TYPE NAME) [--user=username] [--group=groupname] [--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Update the user, group, or service account in a role binding or cluster role binding

Options

- `--all`
Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyang and jsonpath output formats.
- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`
Name of the manager used to track field ownership.
- `--filename, -f`
Filename, directory, or URL to files the resource to update the subjects
- `--group`
Groups to bind to the role
- `--kustomize, -k`
Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`
If true, set subject will NOT contact api-server but run locally.
- `--output, -o`
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive, -R`
Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--serviceaccount`
Service accounts to bind to the role
- `--show-managed-fields`
If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`
Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyang templates [<http://golang.org/pkg/text/template/#pkg-overview>].
- `--user`
Usernames to bind to the role

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

- Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
 - `--cache-dir`
Default cache directory
 - `--certificate-authority`
Path to a cert file for the certificate authority
 - `--client-certificate`
Path to a client certificate file for TLS
 - `--client-key`
Path to a client key file for TLS
 - `--cluster`
The name of the kubeconfig cluster to use
 - `--context`
The name of the kubeconfig context to use
 - `--disable-compression`
If true, opt-out of response compression for all requests to the server
 - `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
 - `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
 - `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
 - `--log-flush-frequency`
Maximum number of seconds between log flushes
 - `--match-server-version`
Require server version to match client version
 - `--namespace, -n`
If present, the namespace scope for this CLI request
 - `--password`
Password for basic authentication to the API server
 - `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
 - `--profile-output`
Name of the file to write the profile to
 - `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
 - `--server, -s`
The address and port of the Kubernetes API server
 - `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
 - `--token`
Bearer token for authentication to the API server
 - `--username`
Username for basic authentication to the API server
 - `--v, -v`
number for the log level verbosity
 - `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text

log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k taint

Usage:

```
d8 k taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]
```

Update the taints on one or more nodes

Options

- `--all`

Select all nodes in the cluster

- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

Name of the manager used to track field ownership.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--overwrite`

If true, allow taints to be overwritten, otherwise reject taint updates that overwrite existing taints.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--validate`

Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if

ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

- Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top

Usage:

```
d8 k top [options]
```

Display resource (CPU/memory) usage

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top node

Aliases: `nodes`, `no`

Usage:

```
d8 k top node [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of nodes

Options

- `--no-headers`
If present, print output without headers
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-capacity`
Print node resources based on Capacity instead of Allocatable(default) of the nodes.
- `--show-swap`
Print node resources related to swap memory.
- `--sort-by`
If non-empty, sort nodes list using specified field. The field can be either 'cpu' or 'memory'.
- `--use-protocol-buffers`
Enables using protocol-buffers to access Metrics API.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`

- Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top pod

Aliases: pods, po

Usage:

```
d8 k top pod [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of pods

Options

- `--all-namespaces, -A`
If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with `--namespace`.
- `--containers`
If present, print usage of containers within a pod.
- `--field-selector`
Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.
- `--no-headers`
If present, print output without headers.
- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. `-l key1=value1,key2=value2,key3 in (value3)`). Matching objects must satisfy all of the specified label constraints.
- `--show-swap`
Print pod resources related to swap memory.
- `--sort-by`
If non-empty, sort pods list using specified field. The field can be either 'cpu' or 'memory'.
- `--sum`
Print the sum of the resource usage
- `--use-protocol-buffers`
Enables using protocol-buffers to access Metrics API.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`

- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k uncordon

Usage:

```
d8 k uncordon NODE [options]
```

Mark node as schedulable

Options

- `--dry-run`
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--selector, -l`
Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`
Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation.
- `--cache-dir`
Default cache directory
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v`, `-v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k version

Usage:

```
d8 k version [options]
```

Print the client and server version information

Options

- `--client`

If true, shows client version only (no server required).

- `--output`, `-o`

One of 'yaml' or 'json'.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k wait

Usage:

```
d8 k wait ([ -f FILENAME] | resource.group/resource.name | resource.group [{"-l label" | --all}]) [--for=create|--for=delete|--for condition=available|--for=jsonpath='{...}'[=value]] [options]
```

Experimental: Wait for a specific condition on one or many resources

Options

- `--all`
Select all resources in the namespace of the specified resource types
- `--all-namespaces, -A`
If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with `--namespace`.
- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to goyaml and jsonpath output formats.

- `--field-selector`

Selector (field query) to filter on, supports '=', '==', and '!='. (e.g. `--field-selector key1=value1,key2=value2`). The server only supports a limited number of field queries per type.

- `--filename, -f`

identifying the resource.

- `--for`

The condition to wait on: `[create|delete|condition=condition-name[=condition-value]]jsonpath='{JSONPath expression}'=[JSONPath value]`. The default condition-value is true. Condition values are compared after Unicode simple case folding, which is a more general form of case-insensitivity.

- `--local`

If true, annotation will NOT contact api-server but run locally.

- `--output, -o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--recursive, -R`

Process the directory used in `-f, --filename` recursively. Useful when you want to manage related manifests organized within the same directory.

- `--selector, -l`

Selector (label query) to filter on, supports '=', '==', and '!='. (e.g. `-l key1=value1,key2=value2`)

- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--template`

Template string or path to template file to use when `-o=go-template, -o=go-template-file`. The template format is goyaml templates [<http://golang.org/pkg/text/template/#pkg-overview>].

- `--timeout`

The length of time to wait before giving up. Zero means check once and don't wait, negative means wait for a week.

Global options

- `--as`

Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

Default cache directory

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
Path to the kuberc file to use for preferences. This can be disabled by exporting `KUBECTL_KUBERC=false` feature gate or turning off the feature `KUBERC=off`.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--match-server-version`
Require server version to match client version
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--profile`
Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
Name of the file to write the profile to
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`
Treat warnings received from the server as errors and exit with a non-zero exit code

d8 mirror

Copy Deckhouse Kubernetes Platform distribution to the local filesystem or third-party registry

d8 mirror pull

Usage:

```
d8 mirror pull <images-bundle-path> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the local filesystem

Options

- `--deckhouse-tag`

Specific Deckhouse build tag to pull. Conflicts with `--since-version` and `--include-platform`. If registry contains release channel image for specified tag, all release channels in the bundle will be pointed to it.

- `--dry-run`

Print what would be pulled without downloading any images. Useful for fast validation of flags and filters.

- `--exclude-module, -e`

Blacklist specific modules from downloading. Format is "module-name[@version]". Use one flag per each module. Overridden by use of `--include-module`."

- `--exclude-package`

Blacklist specific packages from downloading. Format is "package-name[@version]". Use one flag per each package. Overridden by use of `--include-package`.

- `--force`

Overwrite existing bundle packages if they are conflicting with current pull operation.

- `--gost-digest`

Calculate GOST R 34.11-2012 STREEBOG digest for downloaded bundle

- `--ignore-suspend`

Ignore suspended release channels and continue mirroring. Use with caution.

- `--images-bundle-chunk-size, -c`

Split resulting bundle file into chunks of at most N gigabytes

- `--include-module, -i`

Whitelist specific modules for downloading. Use one flag per each module. Disables blacklisting by `--exclude-module`."

Semver constraints (caret, tilde, range) keep only the highest patch in each (major, minor) series, mirroring how platform releases are discovered.

Versions explicitly named with an inclusive boundary operator (`>=` or `<=`) are always preserved — that boundary is part of the user's request and must round-trip even when a newer patch exists in the same minor.

Use the exact-tag form (`=`) when you need a specific older patch unconditionally.

A bare version with no operator (module-name@1.3.0) means "this version or newer within the same major line" and keeps the latest patch per minor. It expands to `>=X.Y.Z <(major+1).0.0`, treating major 0 like any other major: module-name@0.4.0 spans the whole 0.x line (`>=0.4.0 <1.0.0`), NOT just the 0.4 minor as a caret (`^0.4.0 = >=0.4.0 <0.5.0`) would. Prefer this bare form for step-by-step upgrades — it captures every intermediate minor and needs no shell quoting.

Shell note: `>=` and `<=` contain the redirection metacharacters `>` and `<`, so an unquoted module-name@`>=1.3.0` is mangled by bash/zsh (it strips the operator and redirects into a file named `"=1.3.0"`, leaving the module with an empty version). Quote the whole value when you use these operators: `--include-module 'module-name@>=1.3.0'`. The bare-version form above avoids this entirely.

Example:

Available versions for : v1.0.0, v1.1.0, v1.2.0, v1.3.0, v1.3.3, v1.4.0, v1.4.1

module-name@1.3.0 → bare version, expands to `>=1.3.0 <2.0.0` (same major line): keep latest patch per minor — includes v1.3.3 (1.3.x) and v1.4.1 (1.4.x). Versions currently pinned by release channels are pulled in addition.

module-name@~1.3.0 → semver ~ constraint (`>=1.3.0 <1.4.0`): keep latest patch per minor in range — includes v1.3.3. Versions currently pinned by release channels are pulled in addition.

module-name@^1.3.0 → semver ^ constraint (`>=1.3.0 <2.0.0`): keep latest patch per minor —

includes v1.3.3, v1.4.1. For a 0.x version the caret locks the minor ($\wedge 0.4.0 = \geq 0.4.0 < 0.5.0$), so use the bare form instead when you want the whole 0.x line.

module-name@>=1.3.0 → range constraint with explicit >= anchor (quote in shell): keep latest patch per minor AND the named anchor v1.3.0 — includes v1.3.0 (anchor), v1.3.3 (1.3.x latest), v1.4.1 (1.4.x latest).

module-name@>=1.3.0 <=1.4.0 → both anchors honoured (quote in shell): includes v1.3.0, v1.3.3, v1.4.0; v1.4.1 is excluded by the upper bound.

module-name@=v1.3.0 → exact tag match: include only v1.3.0 and publish it to all release channels (alpha, beta, early-access, stable, rock-solid).

module-name@=bobV1 → exact tag match: include only bobV1 and publish it to all release channels (alpha, beta, early-access, stable, rock-solid).

module-name@=v1.3.0+stable → exact tag match: include only v1.3.0 and publish it to stable channel

0.x example — available versions for : v0.4.2, v0.4.4, v0.5.3, v0.6.1, v0.7.2

module-name@0.4.0 → bare version, expands to $\geq 0.4.0 < 1.0.0$: keep latest patch per minor across the whole 0.x line — includes v0.4.4, v0.5.3, v0.6.1, v0.7.2.

- **--include-package**

Whitelist specific packages for downloading. Use one flag per each package. Disables blacklisting by --exclude-package.

Packages are mirrored exactly like modules (same name@version constraint dialect), but live under the packages/ registry segment with their release metadata under packages//version.

- **--include-platform**

Select platform releases to download by a semver constraint expression, using the same dialect as --include-module's version part.

Conflicts with --since-version and --deckhouse-tag.

Semver constraints (caret, tilde, range) keep only the highest patch in each (major, minor) series, mirroring the release-discovery rules used for full pulls.

Versions explicitly named with an inclusive boundary operator ($\geq$ or $\leq$) are always preserved — that boundary is part of the user's request and must round-trip even when a newer patch exists in the same minor.

Use the exact-tag form (=) when you need to pin a specific tag, optionally propagating it to a release channel via the +channel suffix.

Examples (available platform versions: v1.63.x, v1.64.x, v1.65.x, v1.66.x, v1.67.x, v1.68.x, v1.69.x, v1.70.x, v1.71.x):

--include-platform ">=1.64 <=1.68" → bounded range: latest patch per minor in v1.64..v1.68, anchors v1.64.0 and v1.68.0 always preserved if present in the registry.

--include-platform "~1.65.0" → semver ~ constraint ($\geq 1.65.0 < 1.66.0$): latest v1.65.x patch only.

--include-platform "^1.65.0" → semver ^ constraint ($\geq 1.65.0 < 2.0.0$): latest patch per minor starting at v1.65.x.

--include-platform "1.65.0" → bare version, expands to $\geq 1.65.0 < 2.0.0$ (same major line): keep

latest patch per minor starting at v1.65.x. Same result as ^1.65.0 here because platform is always major ≥ 1 ; shorthand kept for parity with `--include-module`.

`--include-platform "=v1.65.3"` → exact-tag pin: only v1.65.3 is pulled and propagated to all default release channels, just like `--deckhouse-tag`.

`--include-platform "=v1.65.3+stable"` → exact-tag pin with channel suffix: only v1.65.3 is pulled (channel propagation matches `--deckhouse-tag`).

- `--insecure`

Interact with registries over HTTP.

- `--installer-tag`

Specific Deckhouse installer build tag to pull. If not specified, the latest tag for the installer will be pulled from the registry path.

- `--license, -l`

Deckhouse license key. Shortcut for `--source-login=license-token --source-password=<>`.

- `--modules-path-suffix`

Suffix to append to source repo path to locate modules.

- `--no-installer`

Do not pull Deckhouse installer into bundle.

- `--no-modules`

Do not pull Deckhouse modules into bundle.

- `--no-packages`

Do not pull Deckhouse packages into bundle.

- `--no-platform`

Do not pull Deckhouse Kubernetes Platform into bundle.

- `--no-pull-resume`

Do not continue last unfinished pull operation and start from scratch.

- `--no-security-db`

Do not pull security databases into bundle.

- `--only-extra-images`

Pull only extra images for modules (additional images like security databases, scanners, etc.) without pulling main module images.

- `--proxy-registry`

Pull from a proxy/caching registry that does not implement the registry catalog API.

Instead of calling the registry's "list tags" endpoint (which proxy registries typically return empty), this mode probes individual tags by incrementing patch -> minor -> major from the version explicitly named via `--include-platform` / `--include-module`. The probe stops once both a new patch and a new minor of the current major fail to resolve, then attempts the next major; if that also fails the probe terminates and downloads what was discovered.

Requires `--include-platform` when platform is not skipped via `--no-platform`, and at least one `--include-module` when modules are not skipped via `--no-modules`. `--exclude-module` and `--no-platform` are respected.

Cannot be combined with `--deckhouse-tag` or `--since-version` (use `--include-platform`'s lower bound instead).

- `--since-version`

Minimal Deckhouse release to pull. Ignored if above current Rock Solid release. Conflicts with `--deckhouse-tag`.

- `--skip-vex-images`

Do not pull VEX images.

- `--source`

Source registry to pull Deckhouse images from.

- `--source-login`
Source registry login.
- `--source-password`
Source registry password.
- `--tls-skip-verify`
Disable TLS certificate validation.
- `--tmp-dir`
Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;
- `--verbose-summary`
List every module and package in the end-of-pull summary with its resolved versions (and VEX count, when present), instead of just the totals. Output only - it does not change what is pulled.

d8 mirror push

Usage:

```
d8 mirror push [images-bundle-path] <registry> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the third-party registry

Options

- `--file`
`Path` to a tar or chunked package to push. May be repeated to push multiple packages. Can be used instead of the bundle directory argument or combined with it.
- `--insecure`
Interact with registries over HTTP.
- `--modules-path-suffix`
Suffix to append to source repo path to locate modules.
- `--registry-login`, `-u`
Username to log into the target registry.
- `--registry-password`, `-p`
Password to log into the target registry.
- `--tls-skip-verify`
Disable TLS certificate validation.
- `--tmp-dir`
Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;

d8 network

Alias: `n`

A group of commands to operate network related tasks in The Deckhouse Ecosystem.

d8 network cni-migration

Usage:

```
d8 network cni-migration [options]
```

A group of commands to switch CNI in the cluster

d8 network cni-migration cleanup

Usage:

```
d8 network cni-migration cleanup [options]
```

Cleans up resources created during CNI switching

d8 network cni-migration switch

Usage:

```
d8 network cni-migration switch [options]
```

Initiates the CNI switching

Options

- `--to-cni`
Target CNI provider to switch to. Supported values: cilium, flannel, simple-bridge

d8 network cni-migration watch

Usage:

```
d8 network cni-migration watch [options]
```

Monitors the CNI switching progress

d8 package

Package build and bootstrap tool for containerized packages

d8 package bootstrap

Usage:

```
d8 package bootstrap [options]
```

Bootstrap a new package (module or application)

d8 package bootstrap application

Usage:

```
d8 package bootstrap application <name> [options]
```

Bootstrap a new application package

Options

- `--extended`
Generate extended files in the package
- `--hooks`
Generate hooks in the package
- `--output, -o`
Custom path for the package (default: /)
- `--werf`
Use werf file for images

d8 package bootstrap module

Usage:

```
d8 package bootstrap module <name> [options]
```

Bootstrap a new module package

Options

- `--extended`
Generate extended files in the package
- `--hooks`
Generate hooks in the package
- `--output, -o`
Custom path for the package (default: /)
- `--werf`
Use werf file for images

d8 package build

Usage:

```
d8 package build --version <version> [--repository <repository>] [--force] [options]
```

Build and push a package to a container registry

Options

- `--debug`
enable debug logging
- `--force, -f`
force update version in registry
- `--repo, -r`
Repository (env: PACKAGE_BUILD_REPOSITORY)
- `--sign`
sign package with certificate and key. --sign-cert and --sign-key are required
- `--sign-cert`
sign certificate path or base64 string (env: PACKAGE_BUILD_SIGN_CERT)
- `--sign-key`
sign key path or base64 string or vault url (env: PACKAGE_BUILD_SIGN_KEY)
- `--token, -t`
Registry token (env: PACKAGE_BUILD_REPOSITORY_TOKEN)
- `--user, -u`
Registry user (env: PACKAGE_BUILD_REPOSITORY_USER)
- `--version, -v`
Package version

d8 package render

Usage:

```
d8 package render [options]
```

Render package templates to Kubernetes manifests

Options

- `--file`
Show only manifests rendered from the template with this file name
- `--render-file`
Write the clean render (without '# Source:' headers) to this file path

d8 package verify

Usage:

```
d8 package verify [options]
```

Verify a package structure and configuration

Options

- `--hide-warnings`
Hide warning-level messages
- `--lint-config`
Path to lint config file
- `--remote, -r`
Verify a package image from a registry reference
- `--show-ignored`
Show ignored-level messages

d8 snapshot

Snapshot operations (create, delete, download, restore, upload, get)

d8 snapshot create

Usage:

```
d8 snapshot create <name> [options]
```

Create a Snapshot of a namespace

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--namespace, -n`
namespace to create the Snapshot in (defaults to the kubeconfig context namespace)
- `--output, -o`
Output format: name|json|yaml
- `--selector, -l`
capture only objects matching this label selector (e.g. app=demo,tier=db); sets spec.resourceSelector.matchLabels
- `--timeout`
timeout for --wait
- `--wait`
wait until the Snapshot reports Ready

d8 snapshot delete

Usage:

```
d8 snapshot delete [name...] [options]
```

Delete one or more Snapshots

Options

- `--all`
delete all Snapshots in the namespace
- `--context`
The name of the kubeconfig context to use
- `--ignore-not-found`
do not error if a named Snapshot does not exist
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)
- `--namespace, -n`

namespace of the Snapshots (defaults to the kubeconfig context namespace)

- `--selector, -l`

delete Snapshots matching this label selector (e.g. app=demo,tier=db)

- `--timeout`

timeout for `--wait`

- `--wait`

wait until the Snapshots are fully deleted

d8 snapshot describe

Usage:

```
d8 snapshot describe [flags] <name> [options]
```

Show the snapshot hierarchy as a tree

Options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

- `--namespace, -n`

snapshot namespace (defaults to the kubeconfig context namespace)

d8 snapshot download

Usage:

```
d8 snapshot download [flags] <snapshot> [options]
```

Download a snapshot to a local directory tree

Options

- `--cleanup`

delete the per-volume DataExport (and its server-side export chain) after each volume completes; `--cleanup=false` leaves them in the cluster for debugging

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

- `--max-parallel-downloads`

global cap on concurrent whole-volume-stream downloads across all nodes (independent of --workers and --per-volume-concurrency)

- `--namespace, -n`

snapshot namespace (required)

- `--node`

restrict download to a single node subtree; format '/' (e.g. --node DemoVirtualDisk/bk-disk-a, --node Snapshot/my-snap); the generated snapshot CR name form (e.g. DemoVirtualDiskSnapshot/nss-child-abc) is still accepted

- `--output, -o`

root output directory (required)

- `--per-volume-concurrency`

maximum parallel chunk/file downloads per volume

- `--ttl`

DataExport TTL (e.g. 2h, 30m)

- `--volume-compression`

volume compression codec (none, zstd); block volumes: data.bin[.]; filesystem volumes: per-file

- compressed entries inside an uncompressed data.tar container
- `--volume-compression-level`
compression level for the selected codec (0 = codec default; ignored when `--volume-compression=none`)
- `--workers`
maximum number of nodes downloaded concurrently

d8 snapshot get

Aliases: `list`, `ls`

Usage:

```
d8 snapshot get [options]
```

List Snapshot resources in the cluster

Options

- `--all-namespaces`, `-A`
list Snapshots across all namespaces
- `--context`
The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)
- `--namespace`, `-n`
namespace to list Snapshots from (defaults to the kubeconfig context namespace)
- `--output`, `-o`
Output format: table|json|yaml

d8 snapshot local

Usage:

```
d8 snapshot local [options]
```

Operate on a locally downloaded snapshot archive (no kubeconfig required)

d8 snapshot local describe

Usage:

```
d8 snapshot local describe <DIR> [options]
```

Show the structure of a locally downloaded snapshot archive as a tree

Options

- `--allow-unauthenticated-legacy`
allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe compatibility mode)

d8 snapshot local get

Usage:

```
d8 snapshot local get <DIR> [options]
```

Print a one-line summary of a locally downloaded snapshot archive

Options

- `--allow-unauthenticated-legacy`
allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe compatibility mode)

compatibility mode)

d8 snapshot restore

Usage:

```
d8 snapshot restore [flags] <snapshot> [options]
```

Restore a snapshot in-place into its namespace

Options

- `--context`
The name of the kubeconfig context to use
- `--dry-run`
validate objects via DryRunAll without persisting; skips `--wait` (use to preflight a restore)
- `--edit`
open resolved manifests in `$KUBE_EDITOR/$EDITOR` (falling back to `vi`) before applying; aborts on non-zero exit, unchanged, or empty content
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)
- `--namespace, -n`
snapshot namespace; also the restore target namespace (required)
- `--no-auto-edit`
disable the one automatic editor session for interactive Kubernetes Invalid (HTTP 422) preflight responses
- `--node`
restrict restore to a single node subtree; format `'/'` (e.g. `--node DemoVirtualDisk/bk-disk-a`); the generated snapshot CR name form (e.g. `DemoVirtualDiskSnapshot/nss-child-abc123`) is still accepted
- `--node-api-version`
disambiguate `--node` by exact API version: `'v1'` for core resources or `'/'` for named groups (requires `--node`)
- `--object`
restrict a `--scope` node restore to a single captured object; format `'/'` (requires `--scope` node)
- `--scope`
restore scope: `'subtree'` (default) compiles the addressed node and its whole subtree; `'node'` compiles only the addressed node itself
- `--timeout`
timeout for the `--wait` Bound check
- `--wait`
wait for manifest PVCs to become Bound; a dormant WaitForFirstConsumer PVC with no selected node, live consumer, or provisioning signal completes without polling

d8 snapshot upload

Alias: `import`

Usage:

```
d8 snapshot upload [flags] [options]
```

Upload a downloaded snapshot archive into a namespace

Options

- `--allow-existing`
downgrade namespace preflight conflict check to a warning (import-mode markers from a prior run are never conflicts regardless of this flag)
- `--allow-unauthenticated-legacy`

allow trusted pre-version archives whose snapshot.yaml metadata is unauthenticated (unsafe; explicit compatibility mode)

- `--context`

The name of the kubeconfig context to use

- `--input, -i`

root archive directory produced by 'd8 snapshot download' (required)

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

- `--namespace, -n`

target namespace to import into (required)

- `--node`

restrict import to a single node subtree; format '/' (e.g. `--node VolumeSnapshot/pvc-1`)

- `--skip-unsupported-fs-entries`

skip unsupported filesystem entries and report them after upload (causes data loss for skipped paths)

- `--timeout`

timeout for per-node readiness/completion waits

- `--ttl`

idle TTL for each data-leaf DataImport (e.g. 2h, 30m); must exceed the importer's provisioning and post-upload completion time

- `--workers`

maximum number of data-leaf volume uploads to run in parallel (fixed cap via `errgroup.SetLimit`; default 5)

d8 status

Get cluster status information

Options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 stronghold

Deckhouse Stronghold commands

d8 stronghold audit

Usage:

```
d8 stronghold [command] [args...] audit [options]
```

Interact with audit devices

d8 stronghold audit disable

Usage:

```
d8 stronghold [command] [args...] audit disable [options]
```

Disables an audit device

d8 stronghold audit enable

Usage:

```
d8 stronghold [command] [args...] audit enable [options]
```


Enables an audit device

d8 stronghold audit list

Usage:

```
d8 stronghold [command] [args...] audit list [options]
```

Lists enabled audit devices

d8 stronghold auth

Usage:

```
d8 stronghold [command] [args...] auth [options]
```

Interact with auth methods

d8 stronghold auth disable

Usage:

```
d8 stronghold [command] [args...] auth disable [options]
```

Disables an auth method

d8 stronghold auth enable

Usage:

```
d8 stronghold [command] [args...] auth enable [options]
```

Enables a new auth method

d8 stronghold auth help

Usage:

```
d8 stronghold [command] [args...] auth help [options]
```

Prints usage for an auth method

d8 stronghold auth list

Usage:

```
d8 stronghold [command] [args...] auth list [options]
```

Lists enabled auth methods

d8 stronghold auth move

Usage:

```
d8 stronghold [command] [args...] auth move [options]
```

Move an auth method to a new path

d8 stronghold auth tune

Usage:

```
d8 stronghold [command] [args...] auth tune [options]
```

Tunes an auth method configuration

d8 stronghold debug

Usage:

```
d8 stronghold [command] [args...] debug [options]
```

Runs the debug command

d8 stronghold delete**Usage:**

```
d8 stronghold [command] [args...] delete [options]
```

Delete secrets and configuration

d8 stronghold events**Usage:**

```
d8 stronghold [command] [args...] events [options]
```

d8 stronghold events subscribe**Usage:**

```
d8 stronghold [command] [args...] events subscribe [options]
```

Subscribe to events

d8 stronghold kv**Usage:**

```
d8 stronghold [command] [args...] kv [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv delete**Usage:**

```
d8 stronghold [command] [args...] kv delete [options]
```

Deletes versions in the KV store

d8 stronghold kv destroy**Usage:**

```
d8 stronghold [command] [args...] kv destroy [options]
```

Permanently removes one or more versions in the KV store

d8 stronghold kv enable-versioning**Usage:**

```
d8 stronghold [command] [args...] kv enable-versioning [options]
```

Turns on versioning for a KV store

d8 stronghold kv get**Usage:**

```
d8 stronghold [command] [args...] kv get [options]
```

Retrieves data from the KV store

d8 stronghold kv list

Usage:

```
d8 stronghold [command] [args...] kv list [options]
```

List data or secrets

d8 stronghold kv metadata

Usage:

```
d8 stronghold [command] [args...] kv metadata [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv metadata delete

Usage:

```
d8 stronghold [command] [args...] kv metadata delete [options]
```

Deletes all versions and metadata for a key in the KV store

d8 stronghold kv metadata get

Usage:

```
d8 stronghold [command] [args...] kv metadata get [options]
```

Retrieves key metadata from the KV store

d8 stronghold kv metadata patch

Usage:

```
d8 stronghold [command] [args...] kv metadata patch [options]
```

Patches key settings in the KV store

d8 stronghold kv metadata put

Usage:

```
d8 stronghold [command] [args...] kv metadata put [options]
```

Sets or updates key settings in the KV store

d8 stronghold kv patch

Usage:

```
d8 stronghold [command] [args...] kv patch [options]
```

Sets or updates data in the KV store without overwriting

d8 stronghold kv put

Usage:

```
d8 stronghold [command] [args...] kv put [options]
```

Sets or updates data in the KV store

d8 stronghold kv rollback

Usage:

```
d8 stronghold [command] [args...] kv rollback [options]
```

Rolls back to a previous version of data

d8 stronghold kv undelete

Usage:

```
d8 stronghold [command] [args...] kv undelete [options]
```

Undeletes versions in the KV store

d8 stronghold lease

Usage:

```
d8 stronghold [command] [args...] lease [options]
```

Interact with leases

d8 stronghold lease lookup

Usage:

```
d8 stronghold [command] [args...] lease lookup [options]
```

Lookup the lease of a secret

d8 stronghold lease renew

Usage:

```
d8 stronghold [command] [args...] lease renew [options]
```

Renews the lease of a secret

d8 stronghold lease revoke

Usage:

```
d8 stronghold [command] [args...] lease revoke [options]
```

Revokes leases and secrets

d8 stronghold list

Usage:

```
d8 stronghold [command] [args...] list [options]
```

List data or secrets

d8 stronghold login

Usage:

```
d8 stronghold [command] [args...] login [options]
```

Authenticate locally

d8 stronghold monitor

Usage:

```
d8 stronghold [command] [args...] monitor [options]
```

Stream log messages from a Stronghold server

d8 stronghold namespace

Usage:

```
d8 stronghold [command] [args...] namespace [options]
```

Interact with namespaces

d8 stronghold namespace create

Usage:

```
d8 stronghold [command] [args...] namespace create [options]
```

Create a new namespace

d8 stronghold namespace delete

Usage:

```
d8 stronghold [command] [args...] namespace delete [options]
```

Delete an existing namespace

d8 stronghold namespace list

Usage:

```
d8 stronghold [command] [args...] namespace list [options]
```

List child namespaces

d8 stronghold namespace lock

Usage:

```
d8 stronghold [command] [args...] namespace lock [options]
```

Lock the API for particular namespaces

d8 stronghold namespace lookup

Usage:

```
d8 stronghold [command] [args...] namespace lookup [options]
```

Look up an existing namespace

d8 stronghold namespace patch

Usage:

```
d8 stronghold [command] [args...] namespace patch [options]
```

Patch an existing namespace

d8 stronghold namespace unlock

Usage:

```
d8 stronghold [command] [args...] namespace unlock [options]
```

Unlock the API for particular namespaces

d8 stronghold operator

Usage:

```
d8 stronghold [command] [args...] operator [options]
```

Perform operator-specific tasks

d8 stronghold operator generate-root**Usage:**

```
d8 stronghold [command] [args...] operator generate-root [options]
```

Generates a new root, DR operation, or recovery token

d8 stronghold operator init**Usage:**

```
d8 stronghold [command] [args...] operator init [options]
```

Initializes a server

d8 stronghold operator key-status**Usage:**

```
d8 stronghold [command] [args...] operator key-status [options]
```

Provides information about the active encryption key

d8 stronghold operator members**Usage:**

```
d8 stronghold [command] [args...] operator members [options]
```

Returns the list of nodes in the cluster

d8 stronghold operator raft**Usage:**

```
d8 stronghold [command] [args...] operator raft [options]
```

Interact with Vault's raft storage backend

d8 stronghold operator raft autopilot**Usage:**

```
d8 stronghold [command] [args...] operator raft autopilot [options]
```

d8 stronghold operator raft autopilot get-config**Usage:**

```
d8 stronghold [command] [args...] operator raft autopilot get-config [options]
```

Returns the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot set-config**Usage:**

```
d8 stronghold [command] [args...] operator raft autopilot set-config [options]
```

Modify the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot state

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot state [options]
```

Displays the state of the raft cluster under integrated storage as seen by autopilot

d8 stronghold operator raft demote

Usage:

```
d8 stronghold [command] [args...] operator raft demote [options]
```

Demotes a voter to a permanent non-voter

d8 stronghold operator raft join

Usage:

```
d8 stronghold [command] [args...] operator raft join [options]
```

Joins a node to the Raft cluster

d8 stronghold operator raft list-peers

Usage:

```
d8 stronghold [command] [args...] operator raft list-peers [options]
```

Returns the Raft peer set

d8 stronghold operator raft promote

Usage:

```
d8 stronghold [command] [args...] operator raft promote [options]
```

Promotes a permanent non-voter to a voter

d8 stronghold operator raft remove-peer

Usage:

```
d8 stronghold [command] [args...] operator raft remove-peer [options]
```

Removes a node from the Raft cluster

d8 stronghold operator raft snapshot

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot [options]
```

Restores and saves snapshots from the Raft cluster

d8 stronghold operator raft snapshot restore

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot restore [options]
```

Installs the provided snapshot, returning the cluster to the state defined in it

d8 stronghold operator raft snapshot save

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot save [options]
```

Saves a snapshot of the current state of the Raft cluster into a file

d8 stronghold operator rekey

Usage:

```
d8 stronghold [command] [args...] operator rekey [options]
```

Generates new unseal keys

d8 stronghold operator rotate

Usage:

```
d8 stronghold [command] [args...] operator rotate [options]
```

Rotates the underlying encryption key

d8 stronghold operator seal

Usage:

```
d8 stronghold [command] [args...] operator seal [options]
```

Seals the Stronghold server

d8 stronghold operator step-down

Usage:

```
d8 stronghold [command] [args...] operator step-down [options]
```

Forces Stronghold to resign active duty

d8 stronghold operator unseal

Usage:

```
d8 stronghold [command] [args...] operator unseal [options]
```

Unseals the Stronghold server

d8 stronghold operator usage

Usage:

```
d8 stronghold [command] [args...] operator usage [options]
```

Lists historical client counts

d8 stronghold patch

Usage:

```
d8 stronghold [command] [args...] patch [options]
```

Patch data, configuration, and secrets

d8 stronghold path-help

Usage:

```
d8 stronghold [command] [args...] path-help [options]
```


Retrieve API help for paths

d8 stronghold pki

Usage:

```
d8 stronghold [command] [args...] pki [options]
```

Interact with Stronghold's PKI Secrets Engine

d8 stronghold pki health-check

Usage:

```
d8 stronghold [command] [args...] pki health-check [options]
```

Check a PKI Secrets Engine mount's health and operational status

d8 stronghold pki issue

Usage:

```
d8 stronghold [command] [args...] pki issue [options]
```

Given a parent certificate, and a list of generation parameters, creates an issuer on a specified mount

d8 stronghold pki list-intermediates

Usage:

```
d8 stronghold [command] [args...] pki list-intermediates [options]
```

Determine which of a list of certificates, were issued by a given parent certificate

d8 stronghold pki reissue

Usage:

```
d8 stronghold [command] [args...] pki reissue [options]
```

Uses a parent certificate and a template certificate to create a new issuer on a child mount

d8 stronghold pki verify-sign

Usage:

```
d8 stronghold [command] [args...] pki verify-sign [options]
```

Check whether one certificate validates another specified certificate

d8 stronghold plugin

Usage:

```
d8 stronghold [command] [args...] plugin [options]
```

Interact with Stronghold plugins and catalog

d8 stronghold plugin deregister

Usage:

```
d8 stronghold [command] [args...] plugin deregister [options]
```

Deregister an existing plugin in the catalog

d8 stronghold plugin info

Usage:

```
d8 stronghold [command] [args...] plugin info [options]
```

Read information about a plugin in the catalog

d8 stronghold plugin list**Usage:**

```
d8 stronghold [command] [args...] plugin list [options]
```

Lists available plugins

d8 stronghold plugin register**Usage:**

```
d8 stronghold [command] [args...] plugin register [options]
```

Registers a new plugin in the catalog

d8 stronghold plugin reload**Usage:**

```
d8 stronghold [command] [args...] plugin reload [options]
```

Reload mounted plugin backend

d8 stronghold plugin reload-status**Usage:**

```
d8 stronghold [command] [args...] plugin reload-status [options]
```

Get the status of an active or recently completed global plugin reload

d8 stronghold policy**Usage:**

```
d8 stronghold [command] [args...] policy [options]
```

Interact with policies

d8 stronghold policy delete**Usage:**

```
d8 stronghold [command] [args...] policy delete [options]
```

Deletes a policy by name

d8 stronghold policy list**Usage:**

```
d8 stronghold [command] [args...] policy list [options]
```

Lists the installed policies

d8 stronghold policy read**Usage:**

```
d8 stronghold [command] [args...] policy read [options]
```

Prints the contents of a policy

d8 stronghold policy write

Usage:

```
d8 stronghold [command] [args...] policy write [options]
```

Uploads a named policy from a file

d8 stronghold print

Usage:

```
d8 stronghold [command] [args...] print [options]
```

Prints runtime configurations

d8 stronghold print token

Usage:

```
d8 stronghold [command] [args...] print token [options]
```

Prints token currently in use

d8 stronghold read

Usage:

```
d8 stronghold [command] [args...] read [options]
```

Read data and retrieves secrets

d8 stronghold secrets

Usage:

```
d8 stronghold [command] [args...] secrets [options]
```

Interact with secrets engines

d8 stronghold secrets disable

Usage:

```
d8 stronghold [command] [args...] secrets disable [options]
```

Disable a secret engine

d8 stronghold secrets enable

Usage:

```
d8 stronghold [command] [args...] secrets enable [options]
```

Enable a secrets engine

d8 stronghold secrets list

Usage:

```
d8 stronghold [command] [args...] secrets list [options]
```

List enabled secrets engines

d8 stronghold secrets move

Usage:

```
d8 stronghold [command] [args...] secrets move [options]
```

Move a secrets engine to a new path

d8 stronghold secrets tune**Usage:**

```
d8 stronghold [command] [args...] secrets tune [options]
```

Tune a secrets engine configuration

d8 stronghold ssh**Usage:**

```
d8 stronghold [command] [args...] ssh [options]
```

Initiate an SSH session

d8 stronghold status**Usage:**

```
d8 stronghold [command] [args...] status [options]
```

Print seal and HA status

d8 stronghold token**Usage:**

```
d8 stronghold [command] [args...] token [options]
```

Interact with tokens

d8 stronghold token capabilities**Usage:**

```
d8 stronghold [command] [args...] token capabilities [options]
```

Print capabilities of a token on a path

d8 stronghold token create**Usage:**

```
d8 stronghold [command] [args...] token create [options]
```

Create a new token

d8 stronghold token lookup**Usage:**

```
d8 stronghold [command] [args...] token lookup [options]
```

Display information about a token

d8 stronghold token renew**Usage:**

```
d8 stronghold [command] [args...] token renew [options]
```

Renew a token lease

d8 stronghold token revoke

Usage:

```
d8 stronghold [command] [args...] token revoke [options]
```

Revoke a token and its children

d8 stronghold transit

Usage:

```
d8 stronghold [command] [args...] transit [options]
```

Interact with Stronghold's Transit Secrets Engine

d8 stronghold transit import

Usage:

```
d8 stronghold [command] [args...] transit import [options]
```

Import a key into the Transit secrets engines.

d8 stronghold transit import-version

Usage:

```
d8 stronghold [command] [args...] transit import-version [options]
```

Import key material into a new key version in the Transit secrets engines.

d8 stronghold unwrap

Usage:

```
d8 stronghold [command] [args...] unwrap [options]
```

Unwrap a wrapped secret

d8 stronghold version

Usage:

```
d8 stronghold [command] [args...] version [options]
```

Prints the Stronghold CLI version

d8 stronghold version-history

Usage:

```
d8 stronghold [command] [args...] version-history [options]
```

Prints the version history of the target Stronghold server

d8 stronghold write

Usage:

```
d8 stronghold [command] [args...] write [options]
```

Write data, configuration, and secrets

d8 system

Aliases: `s`, `p`, `platform`

Operate system options. Aliases: `s`, `p`, `platform`

Options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system collect-debug-info

Usage:

```
d8 system collect-debug-info [flags] > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz [options]
```

Collect debug info.

Options

- `--command-timeout`
Timeout for each individual debug command execution
- `--exclude`
Exclude specific files from the debug archive. Use comma-separated values
- `--list-exclude, -l`
List all files that can be excluded from the debug archive
- `--request-interval`
Minimum interval between debug command executions to avoid overloading the cluster (e.g. 200ms, 500ms, 1s). Zero disables rate limiting (default 0s)

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system edit

Usage:

```
d8 system edit [options]
```

Edit configuration files

Options

- `--editor, -e`
Your favourite editor.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system edit cluster-configuration

Usage:

```
d8 system edit cluster-configuration [options]
```

Edit cluster-configuration.

Options

- `--editor, -e`

Your favourite editor.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system edit provider-cluster-configuration

Usage:

```
d8 system edit provider-cluster-configuration [options]
```

Edit provider-cluster-configuration.

Options

- `--editor, -e`

Your favourite editor.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system edit static-cluster-configuration

Usage:

```
d8 system edit static-cluster-configuration [options]
```

Edit static-cluster-configuration.

Options

- `--editor, -e`

Your favourite editor.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system get

Usage:

```
d8 system get [options]
```

Get configuration files

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system get cluster-configuration

Usage:

```
d8 system get cluster-configuration [options]
```

Get cluster-configuration.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system get provider-cluster-configuration

Usage:

```
d8 system get provider-cluster-configuration [options]
```

Get provider-cluster-configuration.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system get static-cluster-configuration

Usage:

```
d8 system get static-cluster-configuration [options]
```

Get static-cluster-configuration.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system logs

Usage:

```
d8 system logs [options]
```

Inspect deckhouse-controller logs.

Options

- `--follow, -f`
Specify if the logs should be streamed.
- `--since`
Show logs newer than a relative duration like 5s, 2m, or 1h. Only one of since-time / since may be used.
- `--since-time`

Show logs after a specific timestamp, e.g. `--since-time='2025-05-19 12:00:00'`. Only one of `since-time` / `since` may be used.

- `--tail`

Limit the number of lines in the output to the last N lines. Defaults to -1 with no limit.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system module

Usage:

```
d8 system module [options]
```

Operate the Deckhouse Kubernetes Platform modules

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system module apply-now

Usage:

```
d8 system module apply-now <module-name> <version> [options]
```

Apply a module release immediately, bypassing update windows.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system module approve

Usage:

```
d8 system module approve <module-name> <version> [options]
```

Approve a pending module release for Manual update policy.

Global options

- `--context`

The name of the kubeconfig context to use

- `--kubeconfig, -k`

KubeConfig of the cluster. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)

d8 system module disable

Usage:

```
d8 system module disable [options]
```

Disable module.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module enable

Usage:

```
d8 system module enable [options]
```

Enable module.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module list

Usage:

```
d8 system module list [options]
```

List enabled modules.

Options

- `--editor, -e`
Your favourite editor.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module maintenance

Usage:

```
d8 system module maintenance [options]
```

Enable or disable module maintenance mode.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module maintenance disable

Usage:

```
d8 system module maintenance disable <module-name> [options]
```

Disable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module maintenance enable**Usage:**

```
d8 system module maintenance enable <module-name> [options]
```

Enable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module snapshots**Usage:**

```
d8 system module snapshots [options]
```

Dump shapshots.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system module values**Usage:**

```
d8 system module values [options]
```

Dump values.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system package**Usage:**

```
d8 system package [options]
```

Operate the Deckhouse Kubernetes Platform packages

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`

KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system package scan

Usage:

```
d8 system package scan <repository-name> [options]
```

Create a scan task for a PackageRepository

Options

- `--dry-run`
Preview the operation without creating it
- `--name`
Custom name for the PackageRepositoryOperation (auto-generated if not specified)
- `--timeout`
Timeout for the scan operation

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system queue

Usage:

```
d8 system queue [options]
```

Dump queues.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system queue list

Usage:

```
d8 system queue list [options]
```

Dump all queues.

Options

- `--output, -o`
Output format: json|yaml|text.
- `--show-empty, -e`
Show empty queues.
- `--watch, -w`
Watch for changes to the queue list.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 system queue main

Usage:

```
d8 system queue main [options]
```

Dump main queue.

Options

- `--output, -o`
Output format: json|yaml|text.

Global options

- `--context`
The name of the kubeconfig context to use
- `--kubeconfig, -k`
KubeConfig of the cluster. (default is \$KUBECONFIG when it is set, \$HOME/.kube/config otherwise)

d8 tools

Alias: `t`

Various useful tools for operating in The Deckhouse Ecosystem.

d8 tools far-convert

Usage:

```
d8 tools far-convert [options]
```

Converts files with Falco rules to FalcoAuditRules CRD format

d8 tools gostsum

Usage:

```
d8 tools gostsum [options]
```

Calculate Streebog checksum of the data according to the RFC 6986 (GOST R 34.11-2012) specifications

Options

- `--hash-length, -l`
Bit-length variant of the algorithm to use. Only 256 (default) and 512 bit lengths are supported.

d8 tools imagedigest

Usage:

```
d8 tools imagedigest [options]
```

Options

- `--debug, -d`
Enable debug logging
- `--insecure, -i`
Allow insecure connections to registries (skip TLS verification)
- `--json`
Use JSON formatter for output logs

d8 tools imagedigest add

Usage:

```
d8 tools imagedigest add <image> [options]
```

Calculating and adding the image digest to the image metadata according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`
Enable debug logging
- `--insecure`, `-i`
Allow insecure connections to registries (skip TLS verification)
- `--json`
Use JSON formatter for output logs

d8 tools imagedigest calculate

Usage:

```
d8 tools imagedigest calculate <image> [options]
```

Calculating the image digest according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`
Enable debug logging
- `--insecure`, `-i`
Allow insecure connections to registries (skip TLS verification)
- `--json`
Use JSON formatter for output logs

d8 tools imagedigest calculate-from-file

Usage:

```
d8 tools imagedigest calculate-from-file <file> [options]
```

Calculating the file digest according to the GOST standard Streebog (GOST R 34.11-2012). For stdin use '-'

Global options

- `--debug`, `-d`
Enable debug logging
- `--insecure`, `-i`
Allow insecure connections to registries (skip TLS verification)
- `--json`
Use JSON formatter for output logs

d8 tools imagedigest validate

Usage:

```
d8 tools imagedigest validate <image> [options]
```

Validating the image digest in the image metadata calculated according to the GOST standard Streebog (GOST R 34.11-2012)

Options

- `--fix`
Fix GOST Image Digest if it is incorrect

Global options

- `--debug`, `-d`
Enable debug logging
- `--insecure`, `-i`
Allow insecure connections to registries (skip TLS verification)
- `--json`
Use JSON formatter for output logs

d8 tools pki

Usage:

```
d8 tools pki [options]
```

Tools for working with control-plane PKI

d8 tools pki certs

Usage:

```
d8 tools pki certs [options]
```

Manage and inspect control-plane TLS certificates

d8 tools pki certs check

Usage:

```
d8 tools pki certs check [PATH] [options]
```

Check expiration of control-plane certificates

Options

- `--kubeconfig-dir`
Directory containing kubeconfig files (full-scan mode only; defaults to the parent of `--path`)
- `--path`
Directory containing the PKI certificates (used in full-scan mode only)

d8 tools pki certs renew

Usage:

```
d8 tools pki certs renew (all | PATH) [options]
```

Renew control-plane certificates and kubeconfig client certificates

Options

- `--dry-run`
Run all without writing any files
- `--kubeconfig-dir`
Directory containing kubeconfig files (defaults to the parent of `--path`)
- `--path`
Directory containing the PKI certificates and CA files

d8 tools pki certs renew all

Usage:

```
d8 tools pki certs renew (all | PATH) all [options]
```

Renew all control-plane certificates and kubeconfig client certificates

Options

- `--dry-run`
Run all without writing any files
- `--kubeconfig-dir`
Directory containing kubeconfig files (defaults to the parent of `--path`)
- `--path`
Directory containing the PKI certificates and CA files
- `--san`
New IP SAN to add to serving certificates (e.g. the new master IP)

d8 tools sig-migrate

Usage:

```
d8 tools sig-migrate [options]
```

Migrate Kubernetes resources by adding and removing migration annotations

Options

- `--as`
Specify a Kubernetes service account for the kubectl operations (impersonation).
- `--context`
The name of the kubeconfig context to use.
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests. (default is `$KUBECONFIG` when it is set, `$HOME/.kube/config` otherwise)
- `--log-level`
Set the log level (INFO, DEBUG, TRACE). Defaults to DEBUG.
- `--measure-stages`
Print execution time for major migration stages.
- `--object`
Process objects by identifier in format `//`. Use 'clusterwide' namespace for cluster-scoped resources. Resource name must match kubectl api-resources output.
- `--retry`
Retry annotation for previously failed objects.
- `--threads`
Number of worker threads for resource discovery and migration. Values ≤ 0 use default.

d8 v

Alias: `virtualization`

Commands to work with Deckhouse Virtualization Platform.

Options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`

- The name of the kubeconfig cluster to use
- `--context`
- The name of the kubeconfig context to use
- `--disable-compression`
- If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
- If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
- Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
- Maximum number of seconds between log flushes
- `--namespace, -n`
- If present, the namespace scope for this CLI request
- `--password`
- Password for basic authentication to the API server
- `--proxy-url`
- If provided, this URL will be used to connect via proxy
- `--request-timeout`
- The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
- The address and port of the Kubernetes API server
- `--tls-server-name`
- If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
- Bearer token for authentication to the API server
- `--user`
- The name of the kubeconfig user to use
- `--username`
- Username for basic authentication to the API server
- `--v, -v`
- number for the log level verbosity
- `--vmodule`
- comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ansible-inventory

Usage:

```
d8 v [command] [args...] ansible-inventory [options]
```

Generate ansible inventory from virtual machines (ansible inventory script compatible).

Options

- `--host`
- Return variables for specific host
- `--list`
- Return all hosts (default behavior if no arguments)
- `--namespace, -n`
- Namespace to list virtual machines from (overrides kubeconfig context namespace)
- `--output, -o`
- Output format: json, ini, or yaml (default: json)

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v collect-debug-info

Usage:

```
d8 v [command] [args...] collect-debug-info (VirtualMachine) [options]
```

Collect debug information for VM: configuration, events, and logs. Output is written as compressed archive to stdout.

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v console

Usage:

```
d8 v [command] [args...] console (VirtualMachine) [options]
```

Connect to a console of a virtual machine.

Options

- `--timeout`

Duration to wait until console is successfully connected (e.g., 1s, 5m, 300s).

Global options

- `--as`

Username to impersonate for the operation

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--proxy-url`

If provided, this URL will be used to connect via proxy

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

If provided, this name will be used to validate server certificate. If this is not provided, hostname

used to contact the server is used.

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v evict

Usage:

```
d8 v [command] [args...] evict (VirtualMachine) [options]
```

Evict a virtual machine.

Options

- `--create-only, -c`

Set this flag to only create the action without status warnings or notifications.

- `--force, -f`

Set this flag to force the operation.

- `--timeout, -t`

Set this flag to change the timeout.

- `--wait, -w`

Set this flag to wait for the operation to complete.

Global options

- `--as`

Username to impersonate for the operation

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v migrate

Usage:

```
d8 v [command] [args...] migrate (VirtualMachine) [options]
```

Migrate a virtual machine.

Options

- `--create-only, -c`
Set this flag to only create the action without status warnings or notifications.
- `--force, -f`
Set this flag to force the operation.
- `--target-node-name`
Set the target node name for virtual machine migration.
- `--timeout, -t`
Set this flag to change the timeout.
- `--wait, -w`
Set this flag to wait for the operation to complete.

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation

- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v port-forward

Usage:

```
d8 v [command] [args...] port-forward name[.namespace] [protocol/]localPort[:targetPort]... [options]
```

Forward local ports to a virtual machine

Options

- `--address`

--address=: Set this to the address the local ports should be opened on

- `--stdio`

--stdio=true: Set this to true to forward the tunnel to stdout/stdin; Only works with a single port

Global options

- `--as`

Username to impersonate for the operation

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

Path to a client key file for TLS

- `--cluster`

The name of the kubeconfig cluster to use

- `--context`

The name of the kubeconfig context to use

- `--disable-compression`

If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--log-flush-frequency`

Maximum number of seconds between log flushes

- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--proxy-url`

If provided, this URL will be used to connect via proxy

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v restart

Usage:

```
d8 v [command] [args...] restart (VirtualMachine) [options]
```

Restart a virtual machine.

Options

- `--create-only, -c`
Set this flag to only create the action without status warnings or notifications.
- `--force, -f`
Set this flag to force the operation.
- `--timeout, -t`
Set this flag to change the timeout.
- `--wait, -w`
Set this flag to wait for the operation to complete.

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v scp

Usage:

```
d8 v [command] [args...] scp [flags] SOURCE TARGET [options]
```

Secure file CoPy from/to a virtual machine.

Options

- `--identity-file, -i`

Specify a path to a private key used for authenticating to the server; If not provided, the client will try to use the local ssh-agent at \$SSH_AUTH_SOCK

- `--known-hosts`

Set a path to the known_hosts file.

- `--local-ssh`

Use the SSH/SCP client available on your system by using this command as ProxyCommand; Default is false: use embedded SSH client with limited capabilities

- `--local-ssh-opts, -t`

Additional options to be passed to the ssh client if --local-ssh=true is set

- `--port, -p`

Specify a port to connect to

- `--preserve`

Preserves modification times, access times, and modes from the original file.

- `--recursive, -r`

Recursively copy entire directories

Global options

- `--as`

Username to impersonate for the operation

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ssh

Usage:

```
d8 v [command] [args...] ssh [-n|--namespace NAMESPACE] VIRTUAL-MACHINE-NAME [options]
```

Open a SSH connection to a virtual machine.

Options

- `--command, -c`
Specify a command to execute in the VM.
- `--identity-file, -i`
Specify a path to a private key used for authenticating to the server; If not provided, the client will try to use the local ssh-agent at \$SSH_AUTH_SOCK

- `--known-hosts`
Set a path to the known_hosts file.
- `--local-ssh`
Use the SSH/SCP client available on your system by using this command as ProxyCommand;
Default is false: use embedded SSH client with limited capabilities
- `--local-ssh-opts, -t`
Additional options to be passed to the ssh client if --local-ssh=true is set
- `--port, -p`
Specify a port to connect to
- `--username, -l`
Specify user to log into virtual machine; If unassigned, this will be empty and the SSH default will apply

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v start

Usage:

```
d8 v [command] [args...] start (VirtualMachine) [options]
```

Start a virtual machine.

Options

- `--create-only, -c`
Set this flag to only create the action without status warnings or notifications.
- `--force, -f`
Set this flag to force the operation.
- `--timeout, -t`
Set this flag to change the timeout.
- `--wait, -w`
Set this flag to wait for the operation to complete.

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`

If present, the namespace scope for this CLI request

- `--password`

Password for basic authentication to the API server

- `--proxy-url`

If provided, this URL will be used to connect via proxy

- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server, -s`

The address and port of the Kubernetes API server

- `--tls-server-name`

If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

Bearer token for authentication to the API server

- `--user`

The name of the kubeconfig user to use

- `--username`

Username for basic authentication to the API server

- `--v, -v`

number for the log level verbosity

- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v stop

Usage:

```
d8 v [command] [args...] stop (VirtualMachine) [options]
```

Stop a virtual machine.

Options

- `--create-only, -c`

Set this flag to only create the action without status warnings or notifications.

- `--force, -f`

Set this flag to force the operation.

- `--timeout, -t`

Set this flag to change the timeout.

- `--wait, -w`

Set this flag to wait for the operation to complete.

Global options

- `--as`

Username to impersonate for the operation

- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

UID to impersonate for the operation

- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

Path to a client certificate file for TLS

- `--client-key`

- Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`
comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v vnc

Usage:

```
d8 v [command] [args...] vnc VirtualMachine [options]
```

Open a vnc connection to a virtual machine.

Options

- `--address`
--address=127.0.0.1: Setting this will change the listening address of the VNC server. Example: --address=0.0.0.0 will make the server listen on all interfaces.
- `--port`
--port=0: Assigning a port value to this will try to run the proxy on the given port if the port is accessible; If unassigned, the proxy will run on a random port

- `--proxy-only`
`--proxy-only=false`: Setting this true will run only the vnc proxy and show the port where VNC viewers can connect

Global options

- `--as`
Username to impersonate for the operation
- `--as-group`
Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
UID to impersonate for the operation
- `--certificate-authority`
Path to a cert file for the certificate authority
- `--client-certificate`
Path to a client certificate file for TLS
- `--client-key`
Path to a client key file for TLS
- `--cluster`
The name of the kubeconfig cluster to use
- `--context`
The name of the kubeconfig context to use
- `--disable-compression`
If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
Maximum number of seconds between log flushes
- `--namespace, -n`
If present, the namespace scope for this CLI request
- `--password`
Password for basic authentication to the API server
- `--proxy-url`
If provided, this URL will be used to connect via proxy
- `--request-timeout`
The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server, -s`
The address and port of the Kubernetes API server
- `--tls-server-name`
If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`
Bearer token for authentication to the API server
- `--user`
The name of the kubeconfig user to use
- `--username`
Username for basic authentication to the API server
- `--v, -v`
number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

Частые вопросы

На странице представлены часто задаваемые вопросы по настройке и использованию Deckhouse Kubernetes Platform.

Общее

Как принудительно обновить модуль?

Чтобы принудительно применить обновление конкретного модуля, установите аннотацию `modules.deckhouse.io/apply-now: "true"` в соответствующем ресурсе `ModuleRelease`.

Аннотация применяет релиз немедленно, не дожидаясь окна обновлений. Требования из `спес.requirements` при этом сохраняются — если они не выполняются, релиз не будет применен.

Пример установки аннотации для модуля `console`:

```
d8 k annotate mr console-v1.43.3 modules.deckhouse.io/apply-now="true"
```

Для удобства это можно сделать с помощью `d8` CLI (имена модулей и версии автодополняются):

```
d8 system module apply-now console v1.43.3
```

Пример ресурса с установленной аннотацией:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleRelease
metadata:
  name: console-v1.43.3
  annotations:
    modules.deckhouse.io/apply-now: "true"
...
```

Как применить обновление DKP, минуя окна обновлений, canary-release и ручной режим обновлений?

Чтобы применить обновление DKP немедленно, установите в соответствующем ресурсе `DeckhouseRelease` аннотацию `release.deckhouse.io/apply-now: "true"`.

В этом случае будут проигнорированы окна обновления, настройки `canary-release` и режим ручного обновления кластера. Обновление применится сразу после установки аннотации.

Пример команды установки аннотации пропуска окон обновлений для версии `v1.56.2`:

```
d8 k annotate deckhouserelases v1.56.2 release.deckhouse.io/apply-now="true"
```

Пример ресурса с установленной аннотацией пропуска окон обновлений:

```
apiVersion: deckhouse.io/v1alpha1
kind: DeckhouseRelease
metadata:
  annotations:
    release.deckhouse.io/apply-now: "true"
...
```

Что делать, если при включенном VPN контейнер с установщиком не может получить доступ к сети?

Если на компьютере, с которого выполняется `bootstrap` кластера, включен VPN, контейнер с установщиком (включая графический инсталлятор) может потерять доступ к сети. Из-за этого, например, веб-интерфейс графического инсталлятора может не открываться в браузере.

Решить проблему можно одним из следующих способов:

- Отключите VPN на компьютере и перезапустите контейнер с установщиком.
- Если отключить VPN нельзя (например, `bootstrap` кластера выполняется в сети VPN), используйте при запуске контейнера с установщиком параметр `--network host` (**для Docker Desktop на Mac OS параметр доступен, начиная с версии 4.34.0**). Это позволит контейнеру получить доступ к сети.

Пример запуска контейнера с установщиком при включённом VPN, с использованием параметра `--`

network host:

```
docker run --network host --pull=always -it -v "$PWD/config.yml:/config.yml" -v
"$HOME/.ssh:/tmp/.ssh/" -v "$PWD/dhctl-tmp:/tmp/dhctl"
registry.deckhouse.ru/deckhouse/ce/install:early-access bash
```

Пример запуска графического инсталлятора при включённом VPN, с использованием параметра `--network host`:

```
docker run --network host --rm --pull always -v $HOME/.d8installer:$HOME/.d8installer -v
/var/run/docker.sock:/var/run/docker.sock -p 127.0.0.1:8080:8080
registry.deckhouse.ru/deckhouse/installer:latest -r $HOME/.d8installer
```

Как сменить канал обновлений для модуля?

Модуль может быть встроенным в DKP или подключенным из источника модулей (определяется с помощью `ModuleSource`). Встроенные модули имеют общий с DKP релизный цикл и обновляются вместе с DKP. **Канал обновлений встроенного модуля всегда соответствует каналу обновлений DKP.** Модуль, подключаемый из источника, имеет собственный релизный цикл, который не зависит от релизного цикла DKP. **Канал обновлений модуля, подключенного из источника, может быть изменен.**

Далее рассматривается процесс смены канала обновлений для модуля, подключенного из источника.

По умолчанию канал обновлений для модулей наследуется от канала обновлений DKP (указывается в параметре `releaseChannel` `ModuleConfig` `deckhouse`). Подробнее про каналы обновлений — в разделе «Каналы обновлений».

Для модулей из источника канал обновлений задается с помощью `ModuleUpdatePolicy`, который затем *привязывается* к модулю через параметр `updatePolicy` в `ModuleConfig`.

Чтобы сменить канал обновлений у модуля из источника, выполните следующие шаги:

1. Определите политику обновления модуля.

Создайте `ModuleUpdatePolicy`, в котором укажите канал обновлений в параметре `releaseChannel`.

Пример `ModuleUpdatePolicy`:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: my-module-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
```

При необходимости измените режим обновления и настройте окна обновлений.

Убедитесь, что политика создана:

```
d8 k get mup my-module-policy
```

Пример ответа:

NAME	RELEASE CHANNEL	UPDATE MODE
my-module-policy	Alpha	AutoPatch

2. Свяжите политику обновления с модулем.

Укажите имя созданной политики обновления в параметре `updatePolicy` `ModuleConfig` соответствующего модуля.

Для редактирования `ModuleConfig` используйте команду (укажите имя модуля):

```
d8 k edit mc my-module
```

Пример `ModuleConfig`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: my-module
spec:
  enabled: true
  # Имя ModuleUpdatePolicy.
  updatePolicy: my-module-policy

```

При изменении канала обновлений модуля, его установленная версия изменится согласно настроенному режиму обновления.

Чтобы посмотреть текущий канал обновлений модуля и другую информацию о его состоянии в кластере, используйте соответствующий объект `Module`.

Пример команды для получения информации о модуле:

```
d8 k get module my-module -O yaml
```

Используемая политика обновления будет указана в поле `properties.updatePolicy`, текущий канал обновлений — в поле `properties.releaseChannel`. Пример вывода:

```

apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  name: my-module
  # ...
properties:
  # ...
  releaseChannel: Alpha # Канал обновлений модуля.
  updatePolicy: my-module-policy # Политика обновлений модуля.
  version: v1.16.10 # Версия модуля.
  # ...

```

Как сделать так, чтобы внешний модуль обновлялся только на патч-версии?

По умолчанию, если у модуля нет собственной политики обновления, режим обновления и окна наследуются из настроек DKP.

Если для DKP задан режим `AutoPatch`, внешний модуль тоже будет автоматически получать только патч-версии в рамках текущей минорной версии. Переход на новую минорную версию потребует ручного подтверждения. В этом случае отдельно ничего настраивать не нужно.

Подробнее о режимах обновления DKP — в разделе «Настройка обновлений».

Если нужно управлять режимом обновления модуля независимо от DKP, создайте `ModuleUpdatePolicy` с `update.mode: AutoPatch` и привяжите её к модулю через параметр `updatePolicy` в `ModuleConfig`:

1. Создайте политику обновления.

Пример `ModuleUpdatePolicy`:

```

apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: my-update-policy
spec:
  releaseChannel: Stable
  update:
    mode: AutoPatch

```

Убедитесь, что политика создана:

```
d8 k get mup my-update-policy
```

2. Свяжите политику с модулем.

Укажите имя политики в параметре `updatePolicy` `ModuleConfig` модуля:

```
d8 k edit mc module
```

Пример ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module
spec:
  enabled: true
  updatePolicy: my-update-policy
```

В режиме `AutoPatch` патч-версии модуля (например, с `v1.16.1` на `v1.16.2`) применяются автоматически с учётом окон обновлений, если они заданы. Для перехода на новую минорную версию (например, с `v1.16.*` на `v1.17.*`) подтвердите соответствующий `ModuleRelease`:

```
d8 k annotate mr module-v1.17.0 modules.deckhouse.io/approved="true"
```

Или с помощью `d8 CLI`:

```
d8 system module approve module v1.17.0
```

Что делать, если образ модуля не скачался и модуль не переустановился?

В некоторых случаях может возникнуть проблема с автоматическим скачиванием образа и переустановкой модуля. Среди этих случаев:

- Повреждение файловой системы или другие проблемы, из-за которых образ модуля стал невалидным.
- Переключение на использование другого хранилища образов.
- Переключение с одной редакции DKP на другую.

При этом модуль может находиться в статусе `Ready`. А ошибка возникает в подах модуля. Чтобы найти проблемный под используйте команду:

```
d8 k -n d8-<module-name> get pods
```

У проблемного пода будет статус, отличающийся от `Running`.

Для просмотра информации о поде используйте команду:

```
d8 k -n d8-<module-name> describe pod <pod-name>
```

Пример сообщения об ошибке в поде при проблеме со скачиванием образа и переустановкой модуля:

```
Failed to pull image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0a7c968a015dd8d18bb5d54645ddb040ddab1ef4": rpc error: code = NotFound desc = failed to pull and
unpack image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0a7c968a015dd8d18bb5d54645ddb040ddab1ef4": failed to resolve reference ...
```

Чтобы скачать образ и переустановить модуль, с которым возникла проблема:

1. Получите список релизов модуля:

```
d8 k get mr -l module=my-module
```

Пример вывода:

NAME	PHASE	UPDATE POLICY	TRANSITIONTIME	MESSAGE
my-module-v3.7.4	Superseded		5d23h	
my-module-v3.7.5	Deployed		5d23h	

Найдите в списке релиз модуля, развернутый в кластере (он должен иметь статус `Deployed`).

2. Добавьте к развернутому релизу аннотацию `modules.deckhouse.io/reinstall=true`:

```
d8 k annotate mr my-module-v3.7.5 modules.deckhouse.io/reinstall=true
```

После добавления аннотации образ модуля заново скачивается из хранилища образов, модуль валидируется с текущими настройками из `ModuleConfig` и устанавливается в кластер. После успешной переустановки аннотация автоматически удаляется из `ModuleRelease`.

Чтобы убедиться, что переустановка модуля прошла успешно, и все поды работают, используйте команду:

```
d8 k -n d8-<module-name> get pods
```

Все поды модуля должны иметь статус `Running`. Пример:

NAME	READY	STATUS	RESTARTS	AGE
backend-567d6c6cdc-g5qgt	1/1	Running	0	2d2h
frontend-7c8b567759-h8jdf	1/1	Running	0	2d2h
observability-gw-86cf75f5d6-7xljh	1/1	Running	0	2d2h

Кластер и инфраструктура

Что делать в случае проблем с добавлением узла в кластер через Cluster API Provider Static?

Если при добавлении узла в кластер через Cluster API Provider Static (CAPS) он остается в статусе `Pending` или в `Bootstrapping`, выполните следующие действия:

1. Проверьте корректность ключей доступа, указанных в ресурсе `SSHCredentials`. Убедитесь, что имя пользователя и SSH-ключ, указанные в `SSHCredentials`, верны.
2. На узле, с добавлением которого возникла проблема, проверьте наличие в `authorized_keys` публичного ключа, соответствующего приватному из `SSHCredentials`. Пример команды для проверки:

```
cat ~/.ssh/authorized_keys
```

3. Проверьте количество узлов, указанное в `NodeGroup`, в которую должен входить добавляемый узел. Убедитесь, что не превышено максимальное количество узлов.
4. Проверьте статус службы `bashible.service` на узле, с добавлением которого возникла проблема:

```
systemctl status bashible.service
```

Он должен быть в статусе `active (running)`. Если сервис находится в статусе `inactive` или `failed` — служба не запустилась. Это указывает на проблему в процессе настройки.

5. Если указанные выше шаги не помогли устранить проблему, удалите из кластера проблемный узел и его ресурс `StaticInstance`, чтобы система попыталась создать их заново. Для этого:

- Получите список узлов и найдите в нем проблемный:

```
d8 k get nodes
```

- Найдите соответствующий ресурс `StaticInstance`:

```
d8 k get staticinstances -n <namespace-name>
```

- Удалите проблемный узел:

```
d8 k delete node <node-name>
```

- Удалите соответствующий ресурс `StaticInstance`:

```
d8 k delete staticinstances -n <namespace-name> <static-instance-name>
```

Как изменить тип инстанса у узлов с типом CloudPermanent?

Чтобы сменить тип инстанса у узлов с типом `CloudPermanent`, выполните следующие шаги:

1. Сделайте резервную копию `etcd` и папки `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.

4. Убедитесь, что очередь Deckhouse пуста. Для просмотра состояния всех очередей заданий Deckhouse выполните следующую команду:

```
d8 s queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

5. **На локальной машине** запустите контейнер установщика Deckhouse соответствующей редакции и версии (измените адрес container registry при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\.edition}' | tr '[:upper:]'
'[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

6. **В контейнере с инсталлятором** выполните следующую команду, чтобы проверить состояние перед началом работы:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Ответ должен сообщить, что Terraform не нашел расхождений и изменений не требуется.

7. **В контейнере с инсталлятором** выполните команду для редактирования конфигурации кластера (укажите адреса всех master-узлов в параметре `--ssh-host`):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
--ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

8. Отредактируйте параметр `instanceClass` нужной группы узлов, изменив тип инстанса, и сохраните изменения. Пример настроек для `masterNodeGroup` провайдера Yandex Cloud:

```
masterNodeGroup:
  replicas: 3 # требуемое количество master-узлов
  instanceClass:
    cores: 4 # изменить количество CPU
    memory: 8192 # изменить объем памяти (в MB)
    # другие параметры инстанса...
  externalIPAddresses:
    - "Auto" # для каждого master-узла
    - "Auto"
    - "Auto"
```

9. **В контейнере с инсталлятором** выполните следующую команду, чтобы провести обновление узлов:

Внимательно изучите действия, которые планирует выполнить `converge`, когда запрашивает подтверждение.

При выполнении команды узлы будут заменены на новые с подтверждением на каждом узле. Замена будет выполняться по очереди в обратном порядке (2,1,0).

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Следующие действия (п. 9-12) **выполняйте поочередно на каждом** master-узле, начиная с узла с наивысшим номером (с суффиксом 2) и заканчивая узлом с наименьшим номером (с суффиксом 0).

10. **На созданном узле** откройте журнал `systemd-юнита` `bashible.service`. Дождитесь окончания настройки узла — в журнале должно появиться сообщение `nothing to do`:

```
journalctl -fu bashible.service
```

11. Проверьте, что узел etcd отобразился в списке узлов кластера:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

12. Убедитесь, что `control-plane-manager` функционирует на узле.

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
-l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
```

13. Перейдите к обновлению следующего узла.

Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?

При использовании в облачном кластере нескольких групп узлов с разными приоритетами (параметр `spec.cloudInstances.priority`) переключение на заказ узлов из групп с меньшим приоритетом может занимать много времени. Пример такого случая — когда для групп прерываемых узлов (spot, preemptible и т. д.) установлен наивысший приоритет и при недоступности таких инстансов заказ узлов из других групп происходит очень долго.

Для заказа узлов cluster autoscaler по очереди выбирает группу с наивысшим приоритетом, которая **не находится в состоянии backoff**. Backoff — это временная блокировка группы после неудачной попытки заказа узла (например, из-за отсутствия инстансов нужного типа в облаке).

Логика работы при использовании нескольких групп узлов:

1. Cluster autoscaler пытается заказать узел в группе с наивысшим приоритетом.
2. Если в течение времени, заданного в параметре `max-node-provision-time`, узел не появился, попытка считается неудачной.
3. Группа помечается как `failed` и блокируется на время, заданное в параметре `initial-node-group-backoff-duration`.
4. При повторной неудаче для той же группы время блокировки удваивается, но не превышает время, заданное в параметре `max-node-group-backoff-duration`.
5. Cluster autoscaler выбирает следующую по приоритету группу, которая не заблокирована.

Здесь:

Параметр	Значение по умолчанию	Описание
<code>initial-node-group-backoff-duration</code>	5 минут	Начальная длительность блокировки группы после первой неудачи. Удваивается после каждой неудачной попытки заказа узла
<code>max-node-group-backoff-duration</code>	30 минут	Максимальная длительность блокировки группы
<code>max-node-provision-time</code>	15 минут	Время, после которого cluster autoscaler считает попытку заказа узла неудачной

Если для этих параметров установлены значения по умолчанию, переключение на заказ узлов в менее приоритетных группах может занимать значительное время (подробнее — в примере).

Эти параметры можно изменять, чтобы ускорить переключение на заказ узлов в менее приоритетных группах. Чтобы установить нужные значения для параметров `initial-node-group-backoff-duration`, `max-node-group-backoff-duration` и `max-node-provision-time`, отредактируйте объект Deployment `cluster-autoscaler` в неймспейсе `d8-cloud-instance-manager`. Параметры указываются в поле `args` контейнера `cluster-autoscaler`. Пример:


```
...
  containers:
    - name: cluster-autoscaler
      args:
        - --initial-node-group-backoff-duration=1m # Уменьшена начальная длительность
        блокировки группы после первой неудачи.
        - --max-node-provision-time=5m # Уменьшено время, после которого cluster autoscaler
        считает попытку заказа узла неудачной.
  ...
```

Если в кластере с одной группой узлов возникают проблемы с заказом узлов

Если в облачном кластере используется только одна группа узлов (например, только для заказа прерываемых узлов (spot, preemptible)) и при автоматическом масштабировании возникают проблемы с заказом таких узлов, выполните следующие действия:

1. Создайте один или несколько объектов InstanceClass, которые будут использоваться при создании инстансов в кластере. В InstanceClass укажите типы узлов, отличающиеся от указанных в упомянутой выше группе.

i В DKP объекты InstanceClass различаются в зависимости от провайдера. Примеры: AWSInstanceClass, AzureInstanceClass, YandexInstanceClass.

2. Создайте одну или несколько новых групп узлов (объект NodeGroup). В параметре `spec.cloudInstances.classReference` укажите созданные на предыдущем шаге объекты InstanceClass. В параметре `spec.cloudInstances.priority` задайте приоритет группы узлов.
3. При необходимости измените параметры `max-node-provision-time`, `max-node-group-backoff-duration` и `initial-node-group-backoff-duration`, чтобы ускорить переключение на менее приоритетные группы.

Пример процесса заказа узлов с настройками по умолчанию

Предположим, есть 3 группы узлов с разными приоритетами:

- **группа А** (приоритет 50);
- **группа В** (приоритет 30);
- **группа С** (приоритет 0).

В таком случае процесс заказа узлов может быть следующим:

Время	Событие
10:00	Cluster autoscaler пытается заказать узел в А
10:15	Попытка заказа узла в А в течение 15 минут (<code>max-node-provision-time</code>) завершается неудачей → блокировка А на 5 минут (<code>initial-node-group-backoff-duration</code>)
10:15:xx	Cluster autoscaler пытается заказать узел в В (следующая по приоритету, не заблокирована)
10:30	Попытка заказа узла в В в течение 15 минут (<code>max-node-provision-time</code>) завершается неудачей → блокировка В на 5 минут (<code>initial-node-group-backoff-duration</code>)
10:30:xx	Cluster autoscaler снова пытается заказать узел в А , так как время блокировки (значение по умолчанию для <code>initial-node-group-backoff-duration</code> равно 5 минут) закончилось
10:45	Попытка заказа узла в А снова завершается неудачей → блокировка А на 10 минут (<code>initial-node-group-backoff-duration</code> для А удваивается)
...	И так далее, пока время блокировки А и В не превысит 15 минут (<code>max-node-provision-time</code>). До группы С в таком случае очередь может дойти только примерно через 1,5 часа

Что делать, если после добавления объекта не создаются порождаемые им ресурсы?

Если после создания объекта в системе (например, dexAuthenticator) нужные ресурсы не появились,

выполните следующие шаги:

1. Проверьте, есть ли в кластере критические алерты, которые могут блокировать создание нужных объектов. Для этого используйте команду:

```
d8 k get clusteralerts.deckhouse.io
```

Пример вывода:

NAME	ALERT	SEVERITY	AGE
LAST RECEIVED	STATUS		
012f602592aa7a91	K8SSchedulerTargetDown	3	16h 54s
firing			
0836dc893d5ecc65	KubernetesDeploymentReplicasUnavailable	5	15h 62s
firing			
08742f87d62d0063	NTPDaemonOnNodeDoesNotSynchronizeTime	5	16h 46s
firing			
172cfd38d2f7fd19	D8DeckhouseQueueIsHung	7	12h 66s
firing			
1c5705daf731f5cf	D8StrongholdNoActiveNodes	3	16h 55s
firing			
1d2c2f7d69f69f4b	D8DeckhouseIsNotOnReleaseChannel	9	12h 53s
firing			
205a551243d795f3	D8LogShipperAgentNotScheduledInCluster	7	15h 63s
firing			
2e34039aa7a3018e	D8NodeIsNotUpdating	9	12h 47s
firing			
31baf9a70d657275	D8StrongholdClusterNotHealthy	7	16h 55s
firing			

Подробнее об алертах — в разделе «Список алертов».

2. Проверьте очередь заданий Deckhouse:

```
d8 s queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

Если в очереди много необработанных или долго выполняющихся заданий, это может говорить о проблемах.

3. Проанализируйте логи и события DKP:

- Для просмотра логов в реальном времени используйте команду:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

Пример вывода:

```
{ "level": "info", "logger": "addon-operator", "msg": "ConvergeModules task for
OperatorStartup in phase '', trigger is Operator-
Startup", "binding": "ConvergeModules", "event.type": "OperatorStartup", "queue": "main",
"task.flow": "start", "task.id": "fde0eb3b-5c3e-4da6-a0d8-a52f8ae03428", "time": "2025-
11-26T08:29:33Z" }
{ "level": "warn", "logger": "addon-operator.converge-modules", "msg": "ConvergeModules:
functional scheduler not
finished", "binding": "ConvergeModules", "event.type": "OperatorStartup", "queue": "main",
"task.id": "fde0eb3b-5c3e-4da6-a0d8-a52f8ae03428", "time": "2025-11-26T08:29:33Z" }
```

При анализе логов особое внимание обращайте на предупреждения (**WARNING**) и сообщения об ошибках (**ERROR**).

- Для просмотра событий используйте команду:

```
d8 k -n d8-system get events
```

Пример вывода:

LAST SEEN	TYPE	REASON	OBJECT
11m	Warning	Unhealthy	pod/deckhouse-5886c9bd77-vgdbw
Readiness probe failed: HTTP probe failed with statuscode: 500			
7m22s	Normal	SuccessfulDelete	replicaset/deckhouse-5886c9bd77
Deleted pod: deckhouse-5886c9bd77-vgdbw			
7m20s	Normal	Scheduled	pod/deckhouse-6bc5c4494-fw6z
Successfully assigned d8-system/deckhouse-6bc5c4494-fw6z to sandbox1-master-0			
7m20s	Normal	Pulling	pod/deckhouse-6bc5c4494-fw6z
Pulling image "dev-registry.deckhouse.io/sys/deckhouse-oss@sha256:17ac07634e17422df52720264cddec3916ed6985a77782dc8a24fe5352290e6e"			

При анализе событий особое внимание обращайте на те, у которых тип `Warning`.

Deckhouse

Как узнать версию DKP на каждом из каналов обновлений?

Актуальная информация о версиях DKP на разных каналах обновлений доступна на сайте releases.deckhouse.ru.

Как проверить очередь заданий в Deckhouse?

Как посмотреть состояние всех очередей заданий Deckhouse?

Для просмотра состояния всех очередей заданий Deckhouse выполните следующую команду:

```
d8 s queue list
```

Пример вывода (очереди пусты):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

Как посмотреть состояние очереди заданий main?

Для просмотра состояния очереди заданий `main` Deckhouse выполните следующую команду:

```
d8 s queue main
```

Пример вывода (в очереди `main` 38 заданий):

```
Queue 'main': length 38, status: 'run first task'
```

Пример вывода (очередь `main` пуста):

```
Queue 'main': length 0, status: 'waiting for task 0s'
```

Что делать, если DKP не получает обновления из настроенного канала?

- Проверьте, что настроен нужный канал обновлений.
- Проверьте корректность разрешения DNS-имени хранилища образов Deckhouse.

Получите и сравните IP-адреса хранилища образов Deckhouse (`registry.deckhouse.ru`) на одном из узлов и в поде `deckhouse`. Они должны совпадать.

Пример получения IP-адреса хранилища образов Deckhouse на узле:

```
getent ahosts registry.deckhouse.ru
```

Пример вывода:

```
185.193.90.38    STREAM registry.deckhouse.ru
185.193.90.38    DGRAM
185.193.90.38    RAW
```

Пример получения IP-адреса хранилища образов Deckhouse в поде `deckhouse`:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -C deckhouse -- getent ahosts registry.deckhouse.ru
```

Пример вывода:

```
185.193.90.38    STREAM registry.deckhouse.ru
185.193.90.38    DGRAM  registry.deckhouse.ru
```

Если полученные IP-адреса не совпадают, проверьте настройки DNS на узле. В частности, обратите внимание на список доменов в параметре `search` файла `/etc/resolv.conf` (он влияет на разрешение имен в поде `deckhouse`). Если в параметре `search` файла `/etc/resolv.conf` указан домен, в котором настроено разрешение wildcard-записей, это может привести к неверному разрешению IP-адреса хранилища образов Deckhouse (см. пример).

Пример настроек DNS, которые могут привести к ошибкам в разрешении IP-адреса хранилища образов DKP

Далее описан пример, когда настройки DNS приводят к различному результату при разрешении имен на узле и в поде Kubernetes:

- Пример файла `/etc/resolv.conf` на узле:

```
nameserver 10.0.0.10
search company.my
```

i Обратите внимание, что по умолчанию на узле параметр `ndot` равен **1** (`options ndots:1`). Но в подах Kubernetes параметр `ndot` равен **5**. Таким образом, логика разрешения DNS-имен, имеющих в имени 5 точек и менее, различается на узле и в поде.

- В DNS-зоне `company.my` настроено разрешение wildcard-записей `*.company.my` в адрес `10.0.0.100`. То есть любое DNS-имя в зоне `company.my`, для которого нет конкретной записи в DNS, разрешается в адрес `10.0.0.100`.

Тогда с учетом параметра `search`, указанного в файле `/etc/resolv.conf`, при обращении на адрес `registry.deckhouse.ru` на узле система попытается получить IP-адрес для имени `registry.deckhouse.ru` (так как считает его полностью определенным, учитывая настройку по умолчанию параметра `options ndots:1`).

При обращении же на адрес `registry.deckhouse.ru` **из пода** Kubernetes, учитывая параметры `options ndots:5` (используется в Kubernetes по умолчанию) и `search`, система первоначально попытается получить IP-адрес для имени `registry.deckhouse.ru.company.my`. Имя `registry.deckhouse.ru.company.my` разрешится в IP-адрес `10.0.0.100`, так как в DNS-зоне `company.my` настроено разрешение wildcard-записей `*.company.my` в адрес `10.0.0.100`. В результате к хосту `registry.deckhouse.ru` будет невозможно подключиться и скачать информацию о доступных обновлениях Deckhouse.

Поддерживает ли Deckhouse установку на ОС с realtime (rt) и low-latency ядрами?

Deckhouse Kubernetes Platform может устанавливаться на ОС Linux с realtime (rt) и lowlatency ядрами без дополнительной настройки платформы. Работа DKP с такими ядрами была проверена для DKP EE начиная с версии v1.75.7 на следующих ОС и ядрах:

- CentOS 9 Stream, linux-5.14.0-706.el9.x86_64+rt
- RedOS 8, linux-6.12.85-1.red80.x86_64-rt
- Astra Linux 1.7.5, linux-5.15-lowlatency

Как узнать, что для кластера доступна новая версия DKP?

Как только на установленном в кластере канале обновления появляется новая версия DKP:

- Загорается алерт `DeckhouseReleaseIsWaitingManualApproval`, если кластер использует ручной режим обновлений.
- Появляется новый кастомный ресурс `DeckhouseRelease`. Используйте команду `d8 k get deckhouseleases`, чтобы посмотреть список релизов. Если `DeckhouseRelease` новой версии находится в состоянии `Pending`, указанная версия еще не установлена. Возможные причины, при которых `DeckhouseRelease` может находиться в `Pending`:

- Установлен ручной режим обновлений.
- Установлен автоматический режим обновлений и настроены окна обновлений, интервал которых еще не наступил.
- Установлен автоматический режим обновлений, окна обновлений не настроены, но применение версии отложено на случайный период времени из-за механизма снижения нагрузки на репозиторий образов контейнеров. В поле `status.message` ресурса `DeckhouseRelease` будет соответствующее сообщение.
- Установлен параметр `update.notification.minimalNotificationTime` и заданный в нем интервал еще не прошел.

Как заранее получать информацию о предстоящем обновлении?

Получать заранее информацию об обновлении минорных версий DKP на канале обновлений можно следующими способами:

- Настроить ручной режим обновлений. В этом случае при появлении новой версии на канале обновлений загорится алерт `DeckhouseReleaseIsWaitingManualApproval`, и в кластере появится новый кастомный ресурс `DeckhouseRelease`.
- Настроить автоматический режим обновлений и указать минимальное время в параметре `minimalNotificationTime`, на которое будет отложено обновление. В этом случае при появлении новой версии на канале обновлений в кластере появится новый кастомный ресурс `DeckhouseRelease`. Если указать URL в параметре `update.notification.webhook`, через указанный вебхук будет отправлено уведомление об предстоящем обновлении.

Как понять, что обновление прошло успешно?

Если алерт `DeckhouseUpdating` перестал отображаться, это означает, что обновление завершено.

Также вы можете проверить состояние релизов DKP в кластере, используя следующую команду:

```
d8 k get deckhouserelease
```

Пример вывода:

NAME	PHASE	TRANSITIONTIME	MESSAGE
v1.46.8	Superseded	13d	
v1.46.9	Superseded	11d	
v1.47.0	Superseded	4h12m	
v1.47.1	Deployed	4h12m	

Статус `Deployed` говорит о том, что произошло переключение на соответствующую версию, но не гарантирует успешное завершение обновления.

Для проверки успешности обновления выведите состояние пода `deckhouse`:

```
d8 k -n d8-system get pods -l app=deckhouse
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
deckhouse-7844b47bcd-qtbx9	1/1	Running	0	1d

- Если под находится в статусе `Running` и в колонке `READY` указано `1/1`, это означает, что обновление завершилось успешно.
- Если под находится в статусе `Running`, но в колонке `READY` указано `0/1`, это означает, что обновление еще не завершено. Если так продолжается более 20–30 минут, это может говорить о наличии проблем в работе DKP. Необходима диагностика.
- Если под не находится в статусе `Running`, это может говорить о наличии проблем в работе DKP. Необходима диагностика.

Если что-то пошло не так

- Проверьте логи, используя следующую команду:

```
d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
```

- Соберите отладочную информацию и свяжитесь с технической поддержкой DKP.

Как понять, что в кластере идет обновление?

Во время обновления:

- отображается алерт `DeckhouseUpdating`;

- под `deckhouse` находится не в статусе `Ready`. Если под долго не переходит в статус `Ready`, это может говорить о наличии проблем в работе DKP. Необходима диагностика.

Что делать при наличии проблем с обновлением DKP?

Обновление Deckhouse Kubernetes Platform не проходит, один или несколько подов Deckhouse в нерабочем состоянии

Если обновление Deckhouse Kubernetes Platform не проходит, один или несколько подов Deckhouse в неймспейсе `d8-system` находятся в нерабочем состоянии, выполните следующие действия:

1. Проверьте логи Deckhouse с помощью команды:

```
d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
```

При наличии проблем информация о них будет в выводе. При анализе логов особое внимание обращайте на предупреждения (`WARNING`) и сообщения об ошибках (`ERROR`).

2. Проверьте события подов Deckhouse с помощью команды:

```
d8 k -n d8-system describe po -l app=deckhouse | awk '
/^Name:/ {
    pod = $2;
    print "=== " pod " ===";
    in_events = 0
}
/Events:/ {
    in_events = 1;
    next
}
in_events && /^$/ {
    in_events = 0;
    print "---"
}
in_events && !/^Events:/ {
    print $0
}
' | sed '/^---$/N;/^\\n$/D'
```

В событиях подов содержится ключевая информация о проблемах (например, об ошибках планирования, загрузки образов и т.д.). При анализе событий особое внимание обращайте на те, у которых тип `Warning`.

Пример вывода:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Warning	Unhealthy	4m44s (x1918 over 154m)	kubelet	Readiness probe failed: HTTP probe failed with statuscode: 500

Обновление DKP находится в статусе `Release is suspended`

Состояние релиза `Release is suspended` говорит о том, что он был отложен, и на данный момент недоступен (не рекомендуется) для установки. В таком случае предлагается оставаться на последнем доступном релизе, либо на установленном в данный момент (он будет иметь статус `Deployed`).

Для просмотра списка релизов используйте команду:

```
d8 k get deckhouserelases.deckhouse.io
```

Пример вывода:

NAME	PHASE	TRANSITIONTIME	MESSAGE
v1.69.13	Skipped	3h46m	
v1.69.14	Skipped	3h46m	
v1.69.15	Skipped	3h46m	
v1.69.16	Superseded	160m	
v1.70.12	Suspended	49d	Release is suspended
v1.70.13	Skipped	36d	
v1.70.14	Skipped	34d	
v1.70.15	Skipped	28d	
v1.70.16	Skipped	19d	
v1.70.17	Deployed	160m	
v1.71.3	Suspended	14d	Release is suspended

Идентификация и доступ

Что делать при проблемах с применением настроек DexProvider?

Если вы изменили настройки DexProvider в модуле `user-authn` и при этом наблюдается одна из следующих проблем:

- не видно никаких изменений (настройки не применяются);
- при попытке входа в веб-интерфейс платформы с любым типом авторизации возникает ошибка `500 Internal Server Error` без подробного описания.

Выполните следующие действия:

1. Проверьте статус подов деплоймента dex:

```
d8 k -n d8-user-authn get pod
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
dex-5ddb779b7d-6pbhs	2/2	Running	0	20h
kubeconfig-generator-7c46977b9f-5kdmc	1/1	Running	0	20h

Если модуль работает нормально и в DexProvider указана корректная конфигурация, все поды будут в статусе `Running`.

2. Посмотрите логи проблемного пода:

```
d8 k -n d8-user-authn logs dex-<pod-name>
```

На основе информации из логов исправьте конфигурацию в ресурсе DexProvider и дождитесь перезапуска подов dex. В течение нескольких минут поды перезапустятся автоматически, а веб-интерфейс платформы (находится по адресу `console.<ШАБЛОН_ИМЕН_КЛАСТЕРА>`) станет доступен и в нем отразятся внесенные изменения в ресурс DexProvider.

Kubernetes и планировщик

Что делать при высокой нагрузке на API-сервер?

О проблемах с нагрузкой и потреблением памяти API-сервером могут говорить следующие признаки:

- `kubectl (d8)` долго отвечает или не отвечает совсем (команды выполняются медленно или не выполняются);
- в кластере происходит пересоздание подов без явных причин.

При наличии этих признаков выполните следующие действия:

1. Проверьте потребление ресурсов подами API-сервера. Для этого используйте команду:

```
d8 k -n kube-system top po -l component=kube-apiserver
```

Обратите внимание на потребление памяти (`MEMORY`) и `CPU`.

Пример вывода:

NAME	CPU(cores)	MEMORY(bytes)
kube-apiserver-sandbox1-master-0	251m	1476Mi

2. Проверьте метрики в Grafana.

Для просмотра метрик откройте дашборд «Home» → «Dashboards» → «Kubernetes Cluster» → «Control Plane Status». Изучите графики, связанные с API-сервером («Kube-apiserver CPU Usage», «Kube-apiserver Memory Usage», «Kube-apiserver latency» и т.д.).

- Изучите аудит-логи API-сервера, чтобы выявить источник высокого потребления памяти. Одна из частых причин высокого потребления памяти — большое количество запросов.

Как проверить используемую версию Kubernetes?

Для проверки используемой версии Kubernetes выполните команду:

```
d8 k get nodes
```

Пример вывода:

NAME	STATUS	ROLES	AGE	VERSION
frontend-0	Ready	frontend	118d	v1.31.9
master-0	Ready	control-plane,master	118d	v1.31.9
master-1	Ready	control-plane,master	118d	v1.31.9
master-2	Ready	control-plane,master	118d	v1.31.9
system-0	Ready	system	118d	v1.31.9
system-1	Ready	system	118d	v1.31.9
worker-0	Ready	worker	37d	v1.31.9
worker-1	Ready	worker	19d	v1.31.9

Сеть

Поддерживается ли модуль ingress-nginx? Не помешает ли это пройти аудит PCI DSS?

Модуль `ingress-nginx` остаётся в сопровождении Deckhouse Kubernetes Platform на весь срок поддержки платформы и не зависит от статуса upstream-проекта. Команда Deckhouse отслеживает уязвимости в контроллере и связанных компонентах (NGINX, Lua-модули, базовые образы), предоставляет исправления в релизах DKP и обеспечивает план миграции на Gateway API.

Вендором, ответственным за поддержку, выступает «Флант». DKP также проходит регулярную сертификацию ФСТЭК России. Вместе с регулярным сопровождением модуля и управлением уязвимостями это позволяет пройти аудит PCI DSS и соответствует ожиданиям к ответственности вендора за компонент платформы.

История изменений Deckhouse Kubernetes Platform

Версия 1.76

Обратите внимание

- Компонент `fencing-agent` модуля `node-manager` переведён на gossip-протокол (библиотека `memberlist`) для распределённого мониторинга состояния узлов. Это снижает риск ложных перезагрузок worker-узлов при недоступности control plane или высокой нагрузке на API-сервер.
- Исправлена проблема, при которой `cluster-autoscaler` впадал в тупиковое состояние после неудачного создания машины в облаке при ошибках создания машин, отличных от `ResourceExhausted`. Теперь кластер автоматически восстанавливает способность к масштабированию при временных сбоях облачного провайдера.

Подсистема Deckhouse

- Добавлена возможность развертывания кластера DKP с использованием режима `Proxy` для доступа к хранилищу образов.
- `Kubect1` на узлах кластера заменен на алиас `d8 k` (утилита Deckhouse CLI).
- Изменения в `dhctl` (CLI-установщик):
 - Добавлена возможность запускать `dhctl` как отдельный бинарный файл без предварительной загрузки полного образа со всеми зависимостями. Необходимые компоненты загружаются из хранилища образов по ходу выполнения операции, сокращая время загрузки и инициализации.
 - Добавлена валидация публичных SSH-ключей. Некорректные ключи теперь выявляются до этапа, на котором они могут привести к ошибкам подключения к созданным узлам.
 - Исправлена обработка TLS-сертификатов для хранилищ образов. Теперь вместе с пользовательским центром сертификации (CA) используется системный пул сертификатов.
 - Добавлена проверка `InstanceClass` перед установкой на соответствие выбранному облачному

провайдеру. Это позволяет заранее обнаруживать некорректную конфигурацию и останавливать установку до создания объектов в облачной инфраструктуре.

- Исправлена настройка неймспейсов во время бутстрапа. Теперь `dhctl` обновляет настройки даже если неймспейс уже существует.
- Повышена отказоустойчивость мультимастерного режима. Версии Kubernetes теперь корректно определяются при частичной недоступности master-узлов, а kube-apiserver больше не направляет запросы на etcd-узлы в статусе `learner`.

Подсистема Kubernetes & Scheduling

- В ConfigMap `d8-cluster-kubernetes` добавлена информация о поддерживаемых версиях Kubernetes (`supportedVersions`), доступных для обновления и понижения версиях (`availableVersions`), а также о текущей автоматически обновляемой версии (`automaticVersion`).
- Изменения в модуле `descheduler`:
 - Модуль обновлён до версии 0.35.1. Добавлена поддержка фильтрации неймспейсов с помощью механизма `label selector` и дополнительные улучшения. Полный список доработок доступен в истории изменений модуля.
 - В настройки модуля добавлен параметр `deschedulingInterval`. Он позволяет задать частоту запуска цикла перераспределения подов: `Frequent`, `Moderate` или `Rare`.
 - Добавлена стратегия `RemovePodsHavingTooManyRestarts`. Она вытесняет поды, у которых суммарное число перезапусков контейнеров превышает заданный порог, освобождая ресурсы узлов от подов в состоянии `CrashLoopBackOff`.
 - Модуль теперь автоматически использует Kubernetes Metrics API, если в кластере доступна API-группа `metrics.k8s.io`. Это позволяет стратегиям учитывать реальные данные потребления ресурсов.
- Модуль `vertical-pod-autoscaler` обновлён до версии 1.6.0. Ключевые изменения:
 - Режим `InPlaceOrRecreate` теперь используется по умолчанию. В этом режиме VPA пытается изменить `requests` и `limits` пода без его перезагрузки (in-place-обновления).
 - Уменьшена вероятность перезапуска пода при невозможности in-place-обновления (флаг `--in-place-skip-disruption-budget` контроллера), а также, при in-place-обновлениях не учитывается значение `minReplicas`.
- Рекомендации VPA по памяти теперь округляются до 64 MiB. Это делает значения более читаемыми и снижает число перезапусков подов, вызванных небольшими колебаниями рекомендаций.
- Для Kubernetes 1.34 и выше по умолчанию включён feature gate `DRAExtendedResource`, необходимый для работы модулей.
- Добавлен алерт `D8EtcdHighRequestsLatency`, который сообщает о задержке запросов к etcd. Подобные задержки могут приводить к таймаутам API-сервера. Также добавлена соответствующая метрика для дашборда «Control Plane Status».

Подсистема IAM

- Изменения в модуле `user-authn`:
 - Отключён небезопасный OAuth 2.0 Implicit Flow. В этом потоке access-токен передаётся в URL-фрагменте, что повышает риск его утечки (например, через заголовок `Referer`, историю браузера или логи). Интеграции, использующие Authorization Code Flow, не затрагиваются.
 - Добавлена поддержка стандарта аутентификации SAML 2.0 через ресурс `DexProvider`. Кластер DKP теперь можно подключать к таким провайдерам, как AD FS, Okta, Keycloak, OneLogin и Shibboleth.
 - Для SAML-провайдера реализована поддержка refresh-токенов на стороне Dex. Это позволяет `DexAuthenticator` и `kubeconfig-generator` обновлять токены без повторного входа пользователя вручную.
 - Добавлена поддержка механизма Single Logout (SLO) для SAML. Провайдер может завершить сессию пользователя через `/saml/slo/{connector}`.
 - Исправлена проблема с устаревшим кешем на странице входа. В URL-адрес CSS-файла добавлен параметр `cache` с идентификатором сборки Dex.
- При включенном режиме мультитенантности пользователи без прав доступа, настроенных через `ClusterAuthorizationRule`, больше не видят все неймспейсы кластера в веб-интерфейсе Deckhouse. Теперь применяется поведение `deny-by-default`: без явно заданных правил пользователь получает пустой список неймспейсов. Ограничение не касается привилегированных групп (`system:masters`, `kubeadm:cluster-admins`). В документацию модуля `user-authz` добавлен подраздел с примером конфигурации для выдачи прав на все неймспейсы.
- В изолированных проектах добавлена поддержка модуля `service-with-health-checks`. В шаблон

проекта добавлено исключение NetworkPolicy для неймспейса `d8-service-with-healthchecks`, что позволяет агентам модуля выполнять проверки доступности сервисов и корректно работать в условиях сетевой изоляции.

Подсистема Security

- Gatekeeper обновлён до версии 3.22.0, Ratify — до версии 1.4.0. Унифицировано поведение CEL- и Rego-политик, добавлен доступ к неймспейс-контексту в политиках, а также CLI-инструмент для бенчмаркинга политик.
- Настройка `denyVulnerableImages` перенесена из модуля `admission-policy-engine` в `operator-trivy`. Теперь `operator-trivy` является единым владельцем логики запрета запуска уязвимых образов.
- Модуль `cert-manager` обновлён до версии 1.20.0. В новой версии добавлена поддержка Azure DNS Private Zones в DNS01 challenge и новая аннотация `acme.cert-manager.io/http01-ingress-ingressclassname` для переопределения IngressClass в HTTP-01 solvers, а также учтены исправления безопасности.
- В модуле `multitenancy-manager` обновлены уязвимые Go-зависимости `go-jose`, `golang.org/x/crypto`, `golang.org/x/net`, `grpc`, `protobuf` и ряд других.

Подсистема Cluster & Infrastructure

- Изменения в модуле `node-manager`:
 - Исправлена генерация событий draining hook. Исправление улучшает наблюдаемость процесса удаления узлов.
 - Исправлено состояние StaticInstance, из-за которого CAPI мог бесконечно ожидать удаления инфраструктуры. Исправление предотвращает бесконечные циклы очистки и повторной постановки в очередь и позволяет продолжить восстановление или пересоздание машины в обычном режиме.
 - Apiserver-проху переведён с NGINX на нативную Go-реализацию. Это повышает стабильность распределения трафика и ускоряет переподключение при обрыве соединения.
- Изменения в модуле `cloud-provider-dvp`:
 - Добавлена поддержка гибридных кластеров. Статические master-узлы и облачные worker-узлы теперь можно использовать в одном кластере.
 - Повышена надёжность установки и выполнения `converge`. Переработан механизм ожидания готовности ресурсов и добавлен механизм fail-fast для части ошибок при создании инфраструктуры.
 - Для дочерних кластеров, создаваемых через Cluster API, добавлено автоматическое наследование StorageClass по умолчанию из родительского кластера.
 - Исправлена логика обработки жизненного цикла виртуальных машин. Это снижает вероятность возникновения проблем при удалении виртуальных машин и связанных объектов.
 - Добавлена поддержка параметра `masterNodeGroup.instanceClass.additionalDisks`, позволяющего настраивать дополнительные диски для master-узлов.
- Изменения в модуле `cloud-provider-aws`:
 - Логика освобождения узла от нагрузок при завершении Spot Instance перенесена в DKP. Это делает обработку spot-узлов более единообразной.
 - Исправлена обработка регионов, где ещё не поддерживается API DescribeInstanceTopology. Это снижает число ложных ошибок IAM.
- Изменения в модуле `cloud-provider-openstack`:
 - Добавлена поддержка режима `ipMode: Proxy` для `LoadBalancer` в Kubernetes версии 1.32 и старше.
 - Добавлен параметр `csiDriver.fsGroupPolicy` для CSI-драйвера, определяющий, поддерживает ли том изменение владельца и прав доступа перед монтированием.
- Изменения в модуле `cloud-provider-vsphere`:
 - В модуле включён CSI snapshotter, который добавляет возможность создания снимков томов.
 - Исправлено обнаружение зон и datastore. Теперь учитываются только зоны, указанные в конфигурации провайдера.
- В модуле `cloud-provider-azure` добавлена поддержка NVMe-дисков для виртуальных машин Gen2.
- В ресурс GCPIInstanceClass модуля `cloud-provider-gcp` добавлены параметры `enableNestedVirtualization` и `additionalDisks`, позволяющие настраивать узлы для нагрузок, использующих виртуализацию, и сценарии хранения данных.

- В модуле `cloud-provider-yandex` исправлена проблема, из-за которой удаление `externalIPAddresses` могло приводить к ошибке выполнения `converge`.
- Провайдеры DVP, Huawei Cloud и zVirt переведены на контракт Cluster API `v1beta2`.
- Для облачных провайдеров Huawei Cloud и VCD включены проверки политик безопасности и добавлены необходимые ресурсы `SecurityPolicyException` для инфраструктурных компонентов.

Подсистема Network

- Добавлена поддержка обработки входящего трафика через Gateway API с помощью модуля `alb`.
- Изменения в модуле `cni-cilium`:
 - Добавлена поддержка синхронизации conntrack-таблиц между узлами для обеспечения бесшовной живой миграции виртуальных машин.
 - Добавлена поддержка ответов на ICMP-запросы (`ping`) по ExternalIP для сервисов с `type: LoadBalancer` при использовании MetalLB.
- Реализован механизм переключения между поддерживаемыми CNI-плагинами в кластере DKP.
- В модуле `node-local-dns` добавлена возможность отключения резолвинга DNS-запросов по IPv6 с помощью параметра `disableIPv6`.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.75

Обратите внимание

- Добавлена поддержка Kubernetes 1.35 и прекращена поддержка Kubernetes 1.30. В будущих релизах DKP будет прекращена поддержка Kubernetes 1.31. Версия Kubernetes используемая по умолчанию изменена на 1.33 (параметр `kubernetesVersion` установлен в `Automatic`).
- Обновление до версии DKP 1.75 будет заблокировано, если в кластере используется одна из следующих устаревших версий компонентов (их поддержка прекращена):
 - IngressNginxController v1.9;
 - Istio v1.19.
- В DKP 1.75.0–1.75.3 сразу после установки кластера может срабатывать алерт `PodSecurityStandardsViolation`. Это происходит, если у неймспейса `d8-system` отсутствует лейбл `heritage=deckhouse`, из-за чего политики Pod Security Standards могут ошибочно применяться к системным подам DKP. Проблема исправлена в DKP 1.75.4.

В кластерах, установленных на версиях DKP 1.75.0–1.75.3, добавьте необходимый лейбл вручную с помощью следующей команды:

```
d8 k --as=system:sudouser label namespace d8-system heritage=deckhouse
```

- Изменения в поддержке версий Istio:
 - прекращена поддержка Istio 1.19;
 - версия Istio 1.21 признана устаревшей, и её поддержка будет прекращена в будущих версиях DKP. Чтобы обновить Istio, воспользуйтесь инструкцией.
- Начиная с Kubernetes 1.35, на узлах кластера DKP должен использоваться механизм `cgroup v2`. Предыдущая версия (`cgroup v1`) объявлена устаревшей. В случае, если на узле не поддерживается `cgroup v2`, появится алерт `D8NodeCgroupV2NotSupported`. Подробнее о переходе на `cgroup v2` можно почитать в документации Kubernetes.
- Начиная с Kubernetes 1.36, на узлах кластера DKP должен использоваться `container runtime containerd v2`. Поддержка `containerd v1` будет прекращена. В случае, если на узле не поддерживается `containerd v2`, появится алерт `D8NodeContainerdV2NotSupported`.
- Модуль `vertical-pod-autoscaler` обновлен с версии 1.4.1 до 1.5.1. Начиная с версии 1.5.0, режим обновления `updateMode: Auto` объявлен устаревшим. Для пользовательских VPA рекомендуется заменить `updateMode: Auto` на `updateMode: InPlaceOrRecreate`. Подробнее об этом режиме обновления можно почитать в разделе документации про вертикальное масштабирование.
- При выпуске новых сертификатов с использованием модуля `cert-manager` также происходит перевыпуск приватного ключа сертификата. Это необходимо для предотвращения потенциальных рисков безопасности.
- Для подов, обрабатывающих HTTP-01 challenge через ресурсы `Issuer` и `ClusterIssuer`, заданы `resource requests` по умолчанию. Теперь при создании такие поды потребляют часть общей квоты ресурсов кластера и проекта.
- Модуль `operator-trivy` теперь загружается из внешнего источника и развивается в рамках отдельного релизного цикла. Никаких ручных действий для миграции не требуется.

Подсистема Deckhouse

- В ресурс `ModuleSource` добавлен параметр `scanInterval`, позволяющий управлять частотой проверки хранилища образов контейнеров на наличие новых модулей и их версий.
- Добавлен режим выделенного узла с лейблом `node.deckhouse.io/etcd-arbiter`, на котором запускается только `etcd`, без компонентов `control plane`. Это позволяет развернуть отказоустойчивый кластер по схеме 2 master-узла + выделенный `etcd`-узел со сниженными требованиями к ресурсам по сравнению с классической схемой из трёх полноценных master-узлов. Подробнее о настройке режима можно почитать в разделе документации про управление режимом высокой доступности.
- Добавлены новые режимы работы модуля `registry`: **Proxy** (с кешированием образов) и **Local** (для изолированных сред). Новые режимы поддерживаются только для уже развёрнутых статичных кластеров и пока не доступны при развёртывании кластера DKP.
- Параметры доступа к хранилищу образов теперь можно задавать через настройки модуля `deckhouse` при установке кластера DKP. Добавлена возможность развёртывания кластера DKP с использованием режимов `Direct` и `Unmanaged`. Сохранён режим совместимости `Legacy Unmanaged` (через `InitConfiguration`). При отсутствии явной конфигурации режим выбирается автоматически.
- Добавлена проверка используемой оболочки пользователя: если используется не `bash`, утилита `dhctl` теперь выводит понятное сообщение об ошибке.
- Добавлена проверка `StaticInstance` перед установкой DKP по соответствующим `SSHCredentials`, чтобы утилита `dhctl` не сообщала о готовности до фактического завершения развёртывания кластера.
- Расширено тестовое покрытие проверок перед установкой (`preflight checks`) для повышения предсказуемости развёртывания кластера.
- В `bashible` частично сокращено использование `netcat`, что снижает количество срабатываний со стороны систем безопасности.
- Исправлена проблема, из-за которой невозможно было продолжить установку кластера после прерывания процесса установки.
- Исправлена логика удаления объекта `Node` при выполнении `converge`. Теперь узел корректно удаляется из кластера после уничтожения соответствующей виртуальной машины.
- Исправлена работа флага `--skip-resources` при выполнении `dhctl destroy`. Теперь указанные ресурсы корректно пропускаются.

Подсистема Kubernetes & Scheduling

- Новый компонент `update-observer` модуля `control-plane-manager` позволяет отслеживать в реальном времени статус обновления компонентов `control plane` (`kube-apiserver`, `kube-scheduler`, `kube-controller-manager`) и `kubelet` на узлах. Это позволяет оперативно выявлять расхождения и устраняет «слепые зоны» при обновлении Kubernetes.
- Для всех ресурсов VPA, управляемых DKP, режим обновления `updateMode: Auto` автоматически заменён на `updateMode: InPlaceOrRecreate`. Для кластеров с Kubernetes версии 1.32 и ниже автоматически включается экспериментальная функция (feature gate) `InPlacePodVerticalScaling` в компонентах `control plane` (`kube-apiserver`, `kube-scheduler`, `kube-controller-manager`) и `kubelet`. Это позволяет изменять ресурсы подов без их пересоздания.
- Модуль `descheduler` обновлен до версии 0.34. Основные изменения:
 - приоритетное выселение подов — теперь модуль в первую очередь выселяет поды с большим количеством перезапусков, что повышает стабильность кластера при перераспределении нагрузки;
 - поддержка DRA — модуль теперь корректно учитывает и работает с ресурсами, управляемыми через механизм динамического распределения (`Dynamic Resource Allocation`);
 - из стратегий `lowNodeUtilization` и `highNodeUtilization` удалены пороговые значения по умолчанию для `cpu`, `memory` и `pods`. Поведение модуля приведено в соответствие с `upstream-реализацией`.

Подсистема IAM

- Изменено поведение при получении списка неймспейсов: при запросе всех неймспейсов кластера пользователь увидит только те, к которым у него есть доступ, даже если у него нет прав на просмотр полного списка. Изменение направлено на корректную работу мультитенантности и пользовательских интерфейсов кластера.
- В ресурс `ProjectTemplate` модуля `multitenancy-manager` добавлены опции `unmanaged` и `skip-heritage` (задаются аннотациями на ресурсах), позволяющие администраторам управлять степенью контроля DKP над ресурсами:
 - при включении `unmanaged` модуль `multitenancy-manager` не управляет жизненным циклом объектов;
 - при включении `skip-heritage` управление сохраняется, но допускается изменение ресурсов сторонними контроллерами.
- Добавлена возможность выполнять запросы к Kubernetes API, аутентифицируясь через LDAP с

использованием Basic-аутентификации вместо OIDC-токенов (параметр `ldap.enableBasicAuth` в ресурсе `DexProvider`).

- Авторизационный вебхук модуля `user-authz` теперь работает по принципу fail-closed: при его недоступности авторизация запрещена. Для повышения надежности взаимодействие вебхука с `apiserver` осуществляется через сеть узлов кластера.
- В модуле `user-authn` добавлено глобальное ограничение времени жизни пользовательской сессии (параметр `refreshTokenAbsoluteLifetime`). По истечении указанного срока требуется повторная аутентификация.
- В модуле `user-authn` добавлена поддержка аутентификации в DKP с использованием Kerberos (SPNEGO).
- Обновлён список TLS-шифров `kube-apiserver`. Из флага `--tls-cipher-suites` удалены небезопасные шифры `TLS_RSA_WITH_AES_256_GCM_SHA384` и `TLS_RSA_WITH_AES_128_GCM_SHA256` (считаются небезопасными из-за обмена ключами на основе RSA). Добавлены шифры `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256` и `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`.
- Изменён механизм ограничения анонимного доступа к API-серверу. Вместо использования флага `--anonymous-auth=false` и сайдкар-прокси, теперь применяется ресурс `AuthenticationConfiguration`, разрешающий анонимные запросы только к `/livez`, `/readyz` и `/healthz`. Это позволяет эффективнее использовать ресурсы кластера. При использовании публичного доступа к API-серверу через параметр `publishAPI.enabled` анонимный доступ к probe-запросам по-прежнему ограничен.

Подсистема Security

- Исправлена сериализация пустых массивов и объектов в ресурсах `SecurityPolicy` и `OperationPolicy` модуля `admission-policy-engine`. Ранее часть настроек в этих ресурсах могла применяться некорректно.
- Пересмотрена реализация поддержки расширения `ModSecurity`, в результате чего функциональность `OWASP Core Rule Set (CRS)` для `ModSecurity` теперь доступна во всех поддерживаемых версиях `IngressNginxController`.

Подсистема Cluster & Infrastructure

- Изменена стадия жизненного цикла следующих модулей интеграции с облачными провайдерами:
 - модули `cloud-provider-dvp` (DVP), `cloud-provider-vcd` (VMware Cloud Director), `cloud-provider-huaweicloud` (HuaweiCloud) и `cloud-provider-zvirt` (zVirt) теперь в статусе `General Availability` (был — `Experimental`);
 - модуль `cloud-provider-dynamix` (провайдер Basis Dynamix) теперь в статусе `Preview` (был — `Experimental`).
- Добавлена поддержка использования swar-памяти для подов. Вместо немедленного OOM-kill под может временно использовать swar (при наличии выделенных swar-ресурсов на узле) и продолжить работу после пикового потребления ресурсов. Требуется поддержка `cgroup v2` и достаточные ресурсы на узлах. Подробнее про настройку использования swar-памяти можно почитать в описании параметра `kubelet.memorySwap` ресурса `NodeGroup`.
- Для облачного провайдера DVP:
 - Добавлен параметр `networkPolicy`, позволяющий задать правила управления сетевым трафиком к нагрузкам и от нагрузок, запущенных в рамках ресурса `Project`.
 - Добавлены метки `managed-by`, `cluster-uuid` и `vm_name` для всех инфраструктурных объектов.
 - Улучшена диагностика ошибок в CSI-драйвере. CSI-контроллер теперь отображает диагностические сообщения непосредственно в статусе `PersistentVolumeClaim`.
 - Реализован механизм автоматической очистки объектов `VMBDA` при удалении виртуальной машины.
 - Исправлена проблема, при которой объекты `DeckhouseMachine` могли оставаться в состоянии удаления после физического удаления соответствующих виртуальных машин.
 - Устранена проблема, из-за которой `Cloud Controller Manager (CCM)` мог инициировать повторное создание внешних балансировщиков нагрузки после удаления связанного сервиса `Kubernetes`.
- В облачном провайдере `Yandex Cloud` исправлена ошибка отображения, при которой во время установки выводились ложные сообщения о проблемах с NAT в конфигурациях, где NAT фактически не используется.
- В модуле `node-manager`:
 - `Cluster API (CAPI)` обновлен с версии 1.10.6 до 1.11.3. Улучшена стабильность работы и совместимость с новыми провайдерами.
 - Реализован отдельный контроллер для `NodeGroup`-хуков, вынесенный из DKP. Это снижает нагрузку на master-узлы и повышает отзывчивость и информативность интерфейса.
 - Добавлена поддержка покарточной настройки MIG: теперь для каждой физической GPU-карты на узле можно задавать свой набор MIG-партиций. Ранее MIG-конфигурация применялась на уровне всего узла и была одинаковой для всех GPU. Это обеспечивает более гибкое распределение ресурсов под разные типы нагрузок на одном узле.

Подсистема Network

- В IngressNginxController для инлетов HostPort, HostPortWithProxyProtocol, LoadBalancer и LoadBalancerWithProxyProtocol добавлен параметр `acceptClientIPHeadersFrom`. Параметр определяет список подсетей (CIDR), для которых HTTP-заголовки `X-Forwarded-*` считаются доверенными и не перезаписываются при проксировании.
- В модуле `cni-cilium` расширены возможности наблюдаемости сетевого трафика и событий в кластере:
 - добавлены расширенные метрики для мониторинга взаимодействия между заданными объектами (Pods, Services, Namespaces). Метрики настраиваются через параметр `extendedMetrics` ресурса `HubbleMonitoringConfig`;
 - реализован экспорт flow-логов трафика для анализа сетевых взаимодействий между заданными объектами. Экспорт настраивается через параметр `flowLogs` ресурса `HubbleMonitoringConfig`.
- Исправлена проблема, при которой после миграции виртуальной машины Deckhouse Virtualization Platform (DVP) IP-адрес LoadBalancer мог переставать анонсироваться при использовании MetalLB в режиме BGP.
- Изменения в поддержке версий IngressNginxController:
 - прекращена поддержка IngressNginxController v1.9;
 - добавлена поддержка IngressNginxController v1.14.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.74

Обратите внимание

- Добавлена поддержка Kubernetes 1.34 и прекращена поддержка Kubernetes 1.29. В будущих релизах DKP будет прекращена поддержка Kubernetes 1.30. Версия Kubernetes, используемая по умолчанию (параметр `kubernetesVersion` установлен в `Automatic`), изменена на 1.32.

Основные изменения

- Добавлена поддержка управления отдельными экспериментальными и предварительными возможностями Kubernetes (feature gates) с помощью параметра `enabledFeatureGates` модуля `control-plane-manager`. Подробнее об управлении этими возможностями в DKP можно почитать в документации модуля.
- Параметр `--terminated-pod-gc-threshold` (порог удаления выселенных подов) теперь рассчитывается автоматически в зависимости от количества узлов в кластере. Это обеспечивает баланс между стабильностью кластера и возможностями для диагностики. Пороговые значения приведены в документации модуля `control-plane-manager`.
- Максимальное число подов (`maxPods`) для группы узлов теперь устанавливается автоматически на основе параметра `podSubnetNodeCIDRPrefix` ресурса `ClusterConfiguration` (если параметр не задан явно). Это позволяет поддерживать стабильную работу кластера.
- В ресурс `HuaweiCloudInstanceClass` облачного провайдера Huawei Cloud добавлен параметр `vipAddress`, позволяющий задавать виртуальный IP-адрес для всех узлов инстанс-класса.
- В настройках провайдера Huawei Cloud добавлена возможность переопределять главную сеть для основного сетевого интерфейса (параметр `mainNetwork`) и указывать подсети для дополнительных сетевых интерфейсов (параметр `additionalNetworks`) в `CloudPermanent`- и `CloudEphemeral`-узлах.

Важно. При обновлении до DKP 1.74 на Huawei Cloud возможно массовое пересоздание `CloudEphemeral`-узлов. Из-за временной нехватки узлов часть подов может не запуститься.

- Сборка архива с данными для отладки теперь выполняется с помощью утилиты `d8`. При необходимости собрать данные воспользуйтесь инструкцией.
- Теперь при ошибках выполнения хуков сохраняются все собранные ими метрики. Это позволяет не терять отладочную информацию и упрощает последующий анализ причин возникновения ошибки.
- Исправлено зависание очереди Deckhouse в гибридных кластерах, возникавшее при неполном наборе параметров в конфигурации модуля.

Безопасность

- Внедрён контроль целостности модулей на базе файловой системы EROFS для обеспечения безопасности и предотвращения несанкционированных изменений. Для работы новой системы необходима поддержка `dm-verity` и EROFS в ядре.

Сеть

- Добавлена поддержка протокола SCTP (Stream Control Transmission Protocol). Теперь в кластере Deckhouse можно запускать приложения, использующие этот тип соединения.
- В модуль `ingress-nginx` добавлен вспомогательный компонент `geoproxy`. Он нацелен на

повышение стабильности Ingress-контроллера при работе с GeoIP-базами и предоставляет следующие возможности:

- экономия лицензий MaxMind (загрузка баз происходит из одной точки, раз в день);
 - постоянное хранение данных (перезагрузка компонентов не приводит к обращениям на серверы MaxMind);
 - возможность указать своё зеркало для загрузки баз.
- Добавлена метрика Prometheus `bpf_progs_complexity_max_verified_insts` для оценки количества инструкций eBPF-подпрограмм, загружаемых в ядро системы на узлах кластера. Новая метрика позволяет оценить совместимость сетевой подсистемы с имеющимися ядрами. Для работы метрики необходимо наличие ядра Linux версии 5.16 и выше.
 - Добавлена новая метрика `geoip_version` и дашборды для оценки работоспособности GeoIP в кластере.
 - Исправлены проблемы с одновременным обновлением модуля `node-local-dns` совместно с сетевой подсистемой.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.73

Обратите внимание

- Релиз содержит ряд важных изменений, повышающих безопасность. Устранен ряд известных уязвимостей. В частности, устранена уязвимость в модуле `user-authn` (CVE-2025-22868), потенциально позволявшая обходить проверку аутентификации. Рекомендуется запланировать обновление. Подробнее в разделе Безопасность.
- Модуль `dashboard` будет удален в будущих версиях DKP. Используйте веб-интерфейс Deckhouse (необходим включенный модуль `console`).
- Модуль `runtime-audit-engine` теперь загружается из внешнего источника (`ModuleSource deckhouse`).
- В процессе обновления будут перезапущены все компоненты DKP.

Основные изменения

- Добавлена возможность смены настроек `registry` (параметр `mode`) в режиме `Unmanaged` аналогично режиму `Direct`. Также, добавлена настройка режима проверки `registry` при смене настроек (параметр `checkMode`):
 - `Default` — проверяется наличие всех образов для системных модулей и образы `deckhouse-controller`.
 - `Relax` — проверяется наличие только образов `deckhouse-controller`.
- Изменение требований к полю `email` объекта `User`. Адрес `email` должен быть в нижнем регистре. Для уже существующих пользователей сохранена совместимость.
- Обновление Dex до **v2.44.0** теперь позволяет выполнять аутентификацию через подключенные провайдеры, если один из них не работает. Также, аутентификация через провайдера аутентификации теперь может работать через прокси.
- В статусе объекта `User` теперь выводится информация о причине блокировки пользователя (управляется параметром `lockout`).
- В провайдере для интеграции с Deckhouse Virtualization Platform добавлен параметр `additionalDisks` — позволяет создавать и подключать к VM дополнительные диски в `NodeGroup` (задаются `size`, `StorageClass`). Упрощает разнесение данных по нескольким дискам.
- В провайдере для интеграции с VMware Cloud Director добавлена поддержка `additionalMetadata` для объектов (сети, VM, диски). Метаданные объединяются с существующими, значения из `additionalMetadata` имеют приоритет. Изменение `additionalMetadata` приводит к пересозданию групп узлов с типом `CloudEphemeral`, использующих соответствующий `VCDInstanceClass`.
- В провайдере для интеграции с VMware vSphere добавлена возможность указания ID политики хранения SPBM (параметр `storagePolicyID`) и автоматическое создание `StorageClass` для каждой доступной политики хранения SPBM. Теперь можно явно выбирать политику для `master`- и `worker`-узлов, и получать соответствующие классы хранения.
- Добавлены алерты, помогающие запланировать отключение модуля или миграцию:
 - `ModuleIsDeprecated` — сообщает о наличии устаревшего модуля, поддержка которого скоро прекратится.
 - `D8ModuleOutdatedByMajorVersion` — сообщает о том, что модуль отстаёт по мажорным версиям.
- Добавлен алерт `GeoIPDownloadErrorDetected`, сообщающий об ошибках загрузки GeoIP-баз MaxMind.
- Механика оповещений об обновлениях изменена — релиз применяется только при успешной

доставке уведомления на настроенный вебхук. При ошибке доставки применение обновления приостанавливается до восстановления вебхука.

- Изменена структура внутрикластерной документации. Документация всех модулей (включая подключенные) теперь находится в разделе «Модули». Обновлен поиск.
- Для Ingress-контроллера версии 1.10 появилась возможность включать профилировщик (параметр `nginxProfilingEnabled`). Включение профилировщика увеличивает потребление ресурсов, но может потребоваться при поиске причин ошибок в работе контроллера.
- При отправке метрик мониторинга SLA по протоколу Prometheus Remote Write теперь можно установить HTTP-заголовки аутентификации (параметр `headers` ресурса `UpmeterRemoteWrite`).
- Оптимизирована загрузка модулей ядром DKP, снижено время запуска при большом числе модулей.
- В аудит-логах теперь видно, какой OIDC-провайдер выдал токен аутентификации.
- Обновлена версия Deckhouse CLI (`d8`) до v0.23.0:
 - Добавлена команда `d8 status`. Быстрый сводный отчет о состоянии кластера (узлы, релизы, поды Deckhouse, алерты, `registry`, настройки Deckhouse, CNI, состояние очереди).
 - Добавлена команда `d8 k debug`. Запускает встроенный в DKP `debug`-контейнер — образ можно переопределить через `--image`. Упрощает интерактивную отладку подов.
 - Для команды `d8 system queue list` добавлен режим наблюдения за состоянием очереди (ключ `--watch`).

Безопасность

- Повышена базовая безопасность контейнеров. У следующих модулей обновлены профили безопасности для использования только необходимых привилегий и прав доступа: `cni-cilium`, `cni-flannel`, `cni-simple-bridge`, `ingress-nginx`, `istio`, `keepalived`, `kube-dns`, `kube-proxy`, `node-local-dns`, `network-gateway`, `network-policy-engine`, `open-vpn`.
- Добавлен флаг `allowRbacWildcards` политики безопасности (`SecurityPolicy`) позволяющий управлять возможностью использования wildcard при описании объектов `Role` и `RoleBinding` (по умолчанию `true`). Также, политиками безопасности теперь можно ограничивать интерактивные подключения к подам (`CONNECT` для `pods/exec` и `pods/attach`) в пространствах имён.
- Добавлена возможность управления запретом создания подов с `tolerations` из указанного списка (параметр `policies.disallowedTolerations` операционной политики). Это помогает предотвратить попадание пользовательской нагрузки на узлы, отведенные под выделенные задачи.
- Ingress-контроллер обновлен до версии 1.12 — добавлены изменения, повышающие безопасность (`distroless`-образ, устранение уязвимостей и др.).
- Исправлены известные уязвимости в модулях: `operator-trivy`, `registry`, `user-authn`, `cloud-provider-dvp`, `multitenancy-manager`, `admission-policy-engine`, `ingress-nginx`, `alertmanager`, `metallb`, `istio`, `node-local-dns`, `kube-apiserver`.

Сеть

- В провайдере для интеграции с VMware Cloud Director добавлена поддержка `LoadBalancer` на базе VMware NSX Advanced Load Balancer (ALB/Avi) при использовании `NSX-T`. Для работы нужна активированная функция `Load Balancer` на `Edge Gateway`. Если `LoadBalancer` включен после создания кластера, DKP автоматически учтет изменения в течение часа. Для открытых портов создаются пары `Pool + Virtual Service`; при наличии межсетевого экрана необходимо добавить разрешающие правила для внешнего IP-адреса балансировщика и соответствующих портов.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.72

Обратите внимание

- В процессе обновления будут перезапущены все компоненты DKP.
- Теперь, чтобы использовать экспериментальные модули в кластере, необходимо явно дать разрешение (параметр `allowExperimentalModules`). По умолчанию использование экспериментальных модулей отключено. Уже включенные перед обновлением DKP экспериментальные модули отключены не будут. В случае ручного отключения включенного при обновлении DKP экспериментального модуля, для возможности его повторного включения необходимо дать разрешение на использование экспериментальных модулей.
- Если на узлах кластера присутствуют WireGuard-интерфейсы, необходимо обновить ядро Linux до версии не ниже 6.8.

Основные изменения

- Добавлен новый модуль `registry` и возможность менять параметры работы с `container registry` без перезапуска всех компонентов DKP. Теперь доступно два режима работы с `container registry` в DKP: `Unmanaged` (способ, доступный в предыдущих версиях) и `Direct` (новый способ). В режиме `Direct` DKP создает в кластере виртуальный адрес `container registry`, через который осуществляется работа

всех компонентов DKP. Изменение адреса container registry (например, переключение на другой container registry или изменение редакции DKP в кластере) в этом случае не приводит к безусловному перезапуску всех компонентов DKP.

- Добавлена возможность использования рекурсивных DNS-серверов (управляется секцией параметров `recursiveSettings` модуля `cert-manager`), которые применяются для проверки существования записи в DNS перед запуском процесса подтверждения владения доменом методом ACME DNS-01. Это актуально, если один и тот же домен используется как публично, так и внутри кластера, а также, если для домена существуют выделенные авторитетные DNS-серверы.
- Введено разделение модулей на критические и функциональные с помощью флага `critical` в `module.yaml`. Критические модули запускаются первыми, а функциональные — после завершения bootstrap-процесса, при этом их задачи выполняются параллельно и не блокируют очередь в случае сбоя. Это ускоряет установку кластера и повышает отказоустойчивость при запуске модулей.
- Добавлена возможность включения логирования всех DNS-запросов (параметр `enableLogs` модуля `node-local-dns`).
- В модуле `cloud-provider-vcd` добавлена новая схема размещения `WithNAT` для развертывания кластера. Она автоматически настраивает NAT и, при необходимости, правила брандмауэра для доступа к узлам через узел-бастион, а также поддерживает работу как с NSX-T, так и с NSX-V. Это позволяет развертывать кластер в VMware Cloud Director без предварительной настройки окружения (в отличие от схемы размещения `Standard`).

Безопасность

- К событиям аудита (Kubernetes audit log) добавлены поля `user-authn.deckhouse.io/name` и `user-authn.deckhouse.io/preferred_username` для вывода информации о claim пользователя от OIDC-провайдера. Это улучшает контроль и отладку процессов аутентификации.
- Версии Kubernetes 1.30–1.33 обновлены до последних патч-версий.
- Для провайдера AWS добавлена возможность отключить создание групп безопасности по умолчанию (параметр `disableDefaultSecurityGroup`). В этом случае группы безопасности необходимо создать вручную и указать явно в `AWSClusterConfiguration`, `AWSInstanceClass` и `NodeGroup`. Новая возможность расширяет контроль над настройками безопасности.
- Реализована поддержка политики паролей для локальных пользователей (управляется в секции параметров `passwordPolicy`). Теперь можно настраивать минимальный уровень сложности пароля, срок его действия и обязательную ротацию, предотвращать повторное использование старых паролей, а также блокировать аккаунт после заданного числа неудачных попыток входа. Эти изменения позволяют администраторам кластера централизованно применять требования к паролям и повышают уровень безопасности.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.71

Обратите внимание

- Prometheus заменён на Deckhouse Prom++. Если вы хотите продолжать использовать Prometheus, **до обновления платформы** явно выключите модуль `prompp` командой `d8 system module disable prompp`.
- Добавлена поддержка Kubernetes 1.33 и прекращена поддержка Kubernetes 1.28. В будущих релизах DKP поддержка Kubernetes 1.29 будет прекращена. Версия Kubernetes используемая по умолчанию (параметр `kubernetesVersion` установлен в `Automatic`) изменена на 1.31.
- Обновление кластера до версии Kubernetes 1.31 требует последовательного обновления всех узлов с остановкой нагрузки (drain узла). Управлять настройками применения обновлений узла, требующих остановки нагрузки, можно с помощью секции параметров `disruptions`.
- Вместо встроенных модулей `snapshot-controller` и `static-routing-manager` автоматически будут использоваться модули `snapshot-controller` и `static-routing-manager`, загружаемые из внешнего источника (`ModuleSource deckhouse`).
- Новая версия Cilium требует, чтобы ядро Linux на узлах было версии 5.8 или новее. Если на каком-либо из узлов кластера установлено ядро версии ниже 5.8, обновление Deckhouse Kubernetes Platform будет заблокировано. При обновлении поды `cilium` будут перезапущены.
- Все компоненты DKP будут перезапущены в процессе обновления.

Основные изменения

- Добавлена возможность включения обязательного использования двухфакторной аутентификации для статических пользователей. Управляется секцией параметров `staticUsers2FA` модуля `user-authn`.
- Добавлена поддержка GPU на узлах. Доступно управление тремя режимами разделения ресурсов GPU: `Exclusive` (без разделения), `TimeSlicing` (разделение по времени), `MIG` (разделение одного

GPU на несколько экземпляров). Для управления режимом разделения ресурсов GPU используется секция параметров `spec.gpu` в `NodeGroup`. Использование GPU на узле возможно после установки NVIDIA Container Toolkit и драйвера GPU.

- При включении модуля (`d8 system module enable`) или при редактировании ресурса `ModuleConfig`, теперь выводится предупреждение, если для модуля найдено несколько источников модуля. В этом случае требуется явно указать источник модуля в параметре `source` конфигурации модуля.
- Улучшена обработка ошибок конфигурации модулей. Теперь ошибки при работе модуля не блокируют работу DKP, а отображаются в статусах объектов `Module` и `ModuleRelease`.
- Улучшена поддержка виртуализации:
 - Добавлен провайдер интеграции с платформой виртуализации (Deckhouse Virtualization Platform). Новый провайдер позволяет разворачивать кластеры DKP поверх DVP.
 - В модуле `cni-cilium` добавлена поддержка вложенной виртуализации на узлах.
- В модуле `node-manager` добавлены новые возможности для повышения надёжности и управляемости узлов:
 - Добавлена возможность запретить перезапуск узла, если на нем все еще расположены критичные поды (отмеченные лейблом `pod.deckhouse.io/inhibit-node-shutdown`). Это может быть необходимо для обеспечения корректной работы со stateful-нагрузками (например, выполняющих долгую миграцию данных).
 - Добавлена новая версия API `v1alpha2` ресурса `SSHCredential`, в которой параметр `sudoPasswordEncoded` позволяет задавать пароль `sudo` в формате Base64.
 - Параметр `capEmergencyBrake` позволяет отключить CAPI в экстренной ситуации, предотвращая потенциально разрушительные изменения. Поведение аналогично существующей настройке `mcmEmergencyBrake`.
- Добавлена проверка подключения к хранилищу образов контейнеров DKP перед установкой.
- Улучшен механизм ротации файлов при использовании кратковременного хранилища логов (модуль `loki`). Добавлен алерт `LokiInsufficientDiskForRetention`, предупреждающий о нехватке размера хранилища.
- В документацию добавлена справка по командам и параметрам Deckhouse CLI (утилита `d8`).
- При использовании кодирования CEF при сборе логов из Apache Kafka или из сокета, появилась возможность настраивать служебные поля формата, такие как Device Product, Device Vendor и Device ID.
- Поле `passwordHash` в ресурсе `NodeUser` больше не является обязательным. Это позволяет создавать пользователей без пароля, например, в кластерах с внешними системами аутентификации (например, PAM, LDAP).
- Добавлена поддержка CRI Containerd версии 2, использующей CgroupsV2. В новой версии применяется другой формат конфигурации, а также реализован механизм миграции между Containerd V1 и V2. Изменить тип используемого на узлах CRI можно с помощью параметра `cri.type`, а управлять настройками - с помощью параметра `cri.containerdV2`.

Безопасность

- Возможность проверки подписей контейнерных образов стала доступна в редакции DKP SE+. Теперь эта возможность доступна в DKP SE+, EE и CSE.
- Модули `log-shipper`, `deckhouse-controller` и `Istio` (версия 1.21) переведены на distroless-сборку. Это повышает безопасность и обеспечивает более прозрачный и контролируемый процесс сборки.
- Добавлены правила аудита для регистрации взаимодействия с containerd. Отслеживается доступ к сокету `/run/containerd/containerd.sock`, изменение директорий `/etc/containerd`, `/var/lib/containerd` и файла `/opt/deckhouse/bin/containerd`.
- Закрыты известные уязвимости в модулях `loki`, `extended-monitoring`, `operator-prometheus`, `prometheus`, `prometheus-metrics-adapter`, `user-authn`, `cloud-provider-zvirt`, `cloud-provider-dynamix`.

Сеть

- Добавлена поддержка Istio версии 1.25.2. Для этой версии используется оператор Sail вместо Istio Operator, поддержка которого прекращена. Также добавлена поддержка Kiali версии 2.7, без поддержки Ambient Mesh. Версия Istio 1.19 считается устаревшей.
- Добавлена возможность шифрования трафика между узлами и подами с использованием протокола WireGuard (параметр `encryption.mode`).
- Исправлена логика определения готовности сервиса (ресурс `ServiceWithHealthcheck`). Ранее поды без IP-адреса (например, находящиеся в состоянии `Pending`) могли ошибочно попадать в список балансировки.
- Добавлена поддержка алгоритма балансировки нагрузки `least-conn` для сервисов. Алгоритм `least-conn` направляет трафик на бэкэнд сервиса с наименьшим числом активных подключений, что повышает производительность приложений с большим количеством соединений (например, WebSocket). Чтобы управлять алгоритмом балансировки, необходимо включить параметр

`extraLoadBalancerAlgorithmsEnabled` в настройках модуля `cni-cilium`, и использовать на уровне сервиса аннотацию `service.cilium.io/lb-algorithm`, выбрав поддерживаемый алгоритм (`random`, `maglev` или `least-conn`).

- В Cilium 1.17 исправлена ошибка в `cilium-operator`, из-за которой IP-адреса могли не переиспользоваться после удаления `CiliumEndpoint`. Это происходило из-за некорректной очистки фильтра приоритетов, что могло привести к исчерпанию IP-пула в больших кластерах.
- Детализирован список портов, используемых при сетевом взаимодействии:
 - Добавлено и обновлено:
 - 4287/UDP — порт WireGuard для шифрования трафика в CNI Cilium.
 - 4295-4297/UDP — порт используется модулем `cni-cilium` для VXLAN-инкапсуляции трафика между подами при множественной вложенной виртуализации — когда DKP с включенным модулем `virtualization` развернут внутри виртуальных машин, также созданных в DKP с включенным модулем `virtualization`.
 - 4298/UDP — порт используется модулем `cni-cilium` для VXLAN-инкапсуляции трафика между подами, если кластер был развернут на DKP, начиная с версии **1.71** (для кластеров, развернутых на DKP до версии **1.71**, см. примечание для портов 4299/UDP, 8469/UDP и 8472/UDP).
 - 4299/UDP — порт **для кластеров, развернутых на DKP версий 1.64-1.70**. Используется модулем `cni-cilium` для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт, если не включается модуль `virtualization`. **Обратите внимание**, что в таких кластерах включение модуля `virtualization` на DKP до версии 1.70 меняет порт на 4298/UDP.
 - 8469/UDP — порт **для кластеров, развернутых на DKP версии 1.63 и ниже с модулем `virtualization`, включенным до DKP версии 1.63**. Используется модулем `cni-cilium` для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт.
 - 8472/UDP — порт **для кластеров, развернутых на DKP версии 1.63 и ниже**. Используется модулем `cni-cilium` для VXLAN-инкапсуляции трафика между подами. Обновление DKP до более новых версий не изменит занимаемый порт, если не включается модуль `virtualization`. **Обратите внимание**, что в таких кластерах включение модуля `virtualization` на DKP до версии 1.70 меняет порт:
 - включение модуля `virtualization` на DKP версии 1.63 и ниже изменит его на 8469/UDP и не изменит при последующих обновлениях DKP.
 - включение модуля `virtualization` на DKP, начиная с версии 1.64, изменит его на 4298/UDP и не изменит при последующих обновлениях DKP.
 - Удалено:
 - 49152, 49153/TCP — порты использовались для live-миграции виртуальных машин (модуль `virtualization`). Теперь миграция работает через сеть подов.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.70

Обратите внимание

- Модуль `ceph-csi` удален. Вместо него используйте `csi-ceph`. Deckhouse не будет обновляться, если в кластере включен `ceph-csi`. Шаги по миграции с модуля `ceph-csi` приведены в документации модуля `csi-ceph`.
- Добавлена версия 1.12 контроллера Ingress NGINX. Версия контроллера используемая по умолчанию изменена на 1.10. Все Ingress-контроллеры, версия которых не задана явно (параметр `controllerVersion` ресурса `IngressNginxController` или параметр `defaultControllerVersion` модуля `ingress-nginx`), будут перезапущены.
- Удалена метрика `falco_events` (модуль `runtime-audit-engine`). Начиная с DKP 1.68, метрика `falco_events` считалась устаревшей. Используйте метрику `falcosidekick_falco_events_total`. Дашборды и оповещения, основанные на метрике `falco_events`, могут не работать.
- Все компоненты DKP будут перезапущены в процессе обновления.

Основные изменения

- Теперь, в режиме обновления `Auto`, обновление патч-версий (например, с `v1.70.1` на `v1.70.2`) применяется с учетом окон обновлений, если они заданы. Ранее, в этом режиме с учетом окон обновлений применялись только обновления минорных версий (например с `1.69.x` на `1.70.x`), а обновления патч-версий применялись по мере их появления на канале обновлений.
- Добавлена возможность перезагрузки узла, если на соответствующем объекте Node установлена аннотация `update.node.deckhouse.io/reboot`.
- При очистке статического узла теперь удаляются и созданные Deckhouse Kubernetes Platform локальные пользователи.
- Добавлен мониторинг состояния синхронизации Istio в мультикластерной конфигурации. Добавлен алерт мониторинга `D8IstioRemoteClusterNotSynced`, который появляется в следующих случаях:

- удаленный кластер находится в автономном режиме;
 - удаленная конечная точка API недоступна;
 - токен удаленного `ServiceAccount` недействителен или истек;
 - между кластерами существует проблема с TLS или сертификатом.
- Команда `deckhouse-controller collect-debug-info` теперь собирает и отладочную информацию про Istio:
 - ресурсы в пространстве имен `d8-istio`;
 - CRD групп `istio.io` и `gateway.networking.k8s.io`;
 - журналы Istio;
 - журналы Sidecar одного случайно выбранного пользовательского приложения.
- Добавлен дашборд мониторинга с информацией о состоянии сертификатов OpenVPN. По истечении срока действия теперь сертификаты сервера будут перевыпущены, а сертификаты клиентов удалены. Добавлены алерты:
 - `OpenVPNClientCertificateExpired` — о наличии просроченных клиентских сертификатов,
 - `OpenVPNServerCACertificateExpired` — об истечении сертификата OpenVPN CA,
 - `OpenVPNServerCACertificateExpiringSoon` и `OpenVPNServerCACertificateExpiringInAWeek` — о скором окончании срока действия сертификата OpenVPN CA (менее 30 дней или 7 дней до окончания срока действия сертификата соответственно),
 - `OpenVPNServerCertificateExpired` — появляется, если срок действия сертификата сервера OpenVPN истек.
 - `OpenVPNServerCertificateExpiringSoon` и `OpenVPNServerCertificateExpiringInAWeek` — о скором окончании срока действия сертификата сервера OpenVPN (менее 30 дней или 7 дней до окончания срока действия сертификата соответственно).
- Переименованы и изменены дашборды мониторинга:
 - «L2LoadBalancer» переименован в «MetalLB L2». Добавлена фильтрация пулов и колонок;
 - «MetalLB» переименован в «MetalLB BGP». Добавлена фильтрация пулов и колонок. Удалена панель, отвечавшая за отображение ARP-запросов;
 - «L2LoadBalancer / Pools» переименован в «MetalLB / Pools».
- Для модуля `urpmeter` увеличен размер PVC, чтобы вместить данные за период хранения (13 месяцев). В некоторых случаях предыдущего размера PVC было недостаточно.
- В статусе ресурса `ModuleSource` добавлен вывод информации о версии модулей в источнике.
- В статусе ресурса `Module` добавлен вывод информации о стадии жизненного цикла модуля. Модуль в процессе своего жизненного цикла может проходить следующие стадии: `Experimental` (экспериментальная версия), `Preview` (предварительная версия), `General Availability` (общедоступная версия) и `Deprecated` (модуль устарел). Подробнее о жизненном цикле модуля и о том, как понять насколько модуль стабилен, можно узнать в документации.
- Теперь можно выбирать более сильные или современные алгоритмы шифрования (такие как `RSA-3072`, `RSA-4096` или `ECDSA-P256`) для сертификатов `control plane` кластера вместо стандартного `RSA-2048`. Для выбора алгоритма используется параметр `encryptionAlgorithm` ресурса `ClusterConfiguration`.
- Для модуля `descheduler` теперь можно настроить вытеснение подов, использующих локальное хранилище. Для этого используется параметр `evictLocalStoragePods` конфигурации модуля.
- Добавлена возможность управлять уровнем логирования Ingress-контроллера (параметр `controllerLogLevel` ресурса `IngressNginxController`). По умолчанию установлен уровень логирования `Info`. Управление уровнем логирования позволяет, например, предотвратить переполнение сборщика логов при перезапуске Ingress-контроллера.

Безопасность

- Уровень критичности (severity) алертов, сигнализирующих о нарушении политик безопасности, повышен с 7 до 3.
- В провайдерах для `Yandex Cloud`, `Zvirt` и `Basis Dynamix` вместо Terraform теперь используется `OpenTofu`. Это позволит приносить изменения в провайдер, например, чтобы устранять известные уязвимости (CVE).
- Исправлены CVE-уязвимости в модулях: `chrony`, `descheduler`, `dhctl`, `node-manager`, `registry-packages-proxy`, `falco`, `cni-cilium`, `vertical-pod-autoscaler`.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на [GitHub](#).

Версия 1.69

Обратите внимание

- Добавлена поддержка Kubernetes 1.32 и прекращена поддержка Kubernetes 1.27. Версия Kubernetes используемая по умолчанию изменена на 1.30. В будущих релизах DKP поддержка Kubernetes 1.28 будет прекращена.
- Все компоненты DKP будут перезапущены в процессе обновления.

Основные изменения

- Модуль `ceph-csi` считается устаревшим. Запланируйте переход на модуль `csi-ceph`. Подробнее о работе с Ceph читайте в документации.
- Теперь можно выдавать доступ к веб-интерфейсам Deckhouse по именам пользователей с помощью поля `auth.allowedUserEmails`. Разграничение доступа настраивается одновременно с параметром `auth.allowedUserGroups` в конфигурации модулей с веб-интерфейсами: `cilium-hubble`, `dashboard`, `deckhouse-tools`, `documentation`, `istio`, `openvpn`, `prometheus` и `upmeter` (пример для модуля `prometheus`).
- Для модуля `cni-cilium` добавлен дашборд **Cilium Nodes Connectivity Status & Latency** в Grafana, который позволяет отслеживать проблемы с сетевой связностью между узлами. Дашборд отображает матрицу доступности, аналогичную выводу команды `cilium-health status`. Данные собираются из метрик, которые уже доступны в Prometheus.
- Для модуля `control-plane-manager` добавлен алерт `D8KubernetesStaleTokensDetected`, который срабатывает при обнаружении в кластере устаревших токенов сервисных аккаунтов.
- Появилась возможность создавать проект на основе существующего пространства имен, а также включать существующие объекты в проект. Для этого на пространство имен и расположенные в нем объекты необходимо установить аннотацию `projects.deckhouse.io/adopt`. Это позволяет перейти на использование проектов без пересоздания ресурсов в кластере.
- Добавлен статус `Terminating` для ресурсов `ModuleSource` и `ModuleRelease`, который будет отображаться в случае неудачной попытки удаления этих ресурсов.
- В контейнере установщика теперь автоматически настраивается доступ к управлению кластером после успешного развёртывания (генерируется `kubeconfig` в `~/kube/config`, а также настраивается локальный TCP-прокси через SSH-туннель). Это позволяет сразу использовать `kubectrl` локально, без необходимости вручную подключаться к узлу с `control plane` по SSH.
- Теперь изменения в ресурсах Kubernetes при мультикластерных и федеративных конфигурациях отслеживаются напрямую через API Kubernetes. Это ускоряет синхронизацию данных между кластерами и исключает использование устаревших сертификатов. Также теперь полностью исключено мониторирование `ConfigMap` и `Secret` в подах, что устраняет риски, связанные с компрометацией файловой системы внутри контейнеров.
- В CoreDNS добавлен новый плагин `dynamicforward`, который улучшает процесс обработки DNS-запросов в кластере. Плагин интегрируется с модулем `node-local-dns` и постоянно отслеживает изменения на эндпоинтах `kube-dns`, автоматически обновляя список DNS-переадресаторов. Если узел с `control plane` становится недоступен, DNS-запросы продолжают перенаправляться к доступным эндпоинтам, что повышает стабильность работы кластера.
- В модуле `loki` реализована новая стратегия ротации логов. Теперь старые записи удаляются автоматически при достижении порога использования диска. Порог рассчитывается как минимум из двух значений: 95% размера PVC или размер PVC за вычетом объёма, необходимого для хранения двух минут логов при заданной скорости приёма (`ingestionRateMB`). Параметр `retentionPeriodHours` больше не управляет сроком хранения данных и используется только для алертов мониторинга. Если `loki` начнёт удалять записи до истечения заданного периода, пользователю будет отправлен алерт `LokiRetentionPerionViolation` и будет необходимо уменьшить значение `retentionPeriodHours`, либо увеличить размер PVC.
- Добавлен параметр `nodeDrainTimeoutSecond`, который позволяет задать максимальную продолжительность попыток выполнения операции `drain` на узле (в секундах) для каждого ресурса `NodeGroup`. Ранее можно было использовать либо вариант по умолчанию (10 минут), либо уменьшить время до 5 минут (параметр `quickShutdown`, который теперь следует считать устаревшим).
- В настройках модуля `openvpn` теперь можно указать значение нового параметра `defaultClientCertExpirationDays`, который задаёт срок действия клиентских сертификатов.

Безопасность

- Закрыты известные уязвимости в модулях `ingress-nginx`, `istio`, `prometheus` и `local-path-provisioner`.

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Версия 1.68

Обратите внимание

- После обновления у всех источников данных (`DataSource`) Grafana, созданных с помощью ресурса `GrafanaAdditionalDatasource`, изменится UID. Если на источник данных ссылались по UID, то такая связь нарушится.

Основные изменения

- Новый параметр `iamNodeRole` для провайдера AWS. Параметр позволяет задать имя IAM-роли, которая будет привязана ко всем AWS-инстансам узлов кластера. Это может потребоваться, если в

IAM-роль узла нужно добавить больше прав (например, доступ к ECR и т.п.)

- Ускорено создание узлов с типом CloudPermanent. Теперь все такие узлы создаются параллельно. Ранее параллельно создавались CloudPermanent-узлы только в рамках одной группы.
- Изменения в мониторинге:
 - добавлена возможность мониторинга сертификатов в секретах типа `Opaque`;
 - добавлена возможность мониторинга образов в Amazon ECR;
 - исправлена ошибка, из-за которой при перезапуске экземпляров Prometheus могла потеряться часть метрик.
- При использовании мультикластерной конфигурации Istio или федерации, теперь можно явно указать список адресов ingressgateway, которые нужно использовать для организации межкластерных запросов. Ранее эти адреса вычислялись только автоматически, но в некоторых конфигурациях их определить невозможно.
- У аутентификатора (ресурс DexAuthenticator) появился параметр `highAvailability`, который управляет режимом высокой доступности. В режиме высокой доступности запускается несколько реплик аутентификатора. Ранее режим высокой доступности всех аутентификаторов определялся настройками глобального параметра или настройками модуля `user-authn`. Все аутентификаторы, развернутые самим DKP, теперь наследуют режим высокой доступности соответствующего модуля.
- Лейблы узлов теперь можно добавлять, удалять и изменять, используя файлы, хранящиеся на узле в директории `/var/lib/node labels` и её поддиректориях. Полный набор применённых лейблов хранится в аннотации `node.deckhouse.io/last-applied-local-labels`.
- Добавлена поддержка облачного провайдера Huawei Cloud.
- Новый параметр `keepDeletedFilesOpenedFor` в модуле `log-shipper` позволяет настроить период, в течение которого будут храниться открытыми удалённые файлы логов. Опция позволит какое-то время читать логи удалённых подов в случае недоступности хранилища логов.
- TLS-шифрование для сборщиков логов (Elasticsearch, Vector, Loki, Splunk, Logstash, Socket, Kafka) теперь можно настроить, используя секреты, вместо хранения сертификатов в ресурсах `ClusterLogDestination`. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.
- В статусе проекта в разделе `resources` теперь можно увидеть, какие ресурсы проекта были установлены. Такие ресурсы будут отмечены флагом `installed: true`.
- В инсталлятор добавлен параметр `--tf-resource-management-timeout`, позволяющий управлять таймаутом создания ресурсов в облаках. По умолчанию таймаут составляет 10 минут. Параметр имеет влияние только для следующих облаков: AWS, Azure, GCP, Yandex Cloud, OpenStack, Basis Dynamix.

Безопасность

Закрываются известные уязвимости в следующих модулях:

- admission-policy-engine
- chrony
- cloud-provider-azure
- cloud-provider-gcp
- cloud-provider-openstack
- cloud-provider-yandex
- cloud-provider-zvirt
- cni-cilium
- control-plane-manager
- extended-monitoring
- descheduler
- documentation
- ingress-nginx
- istio
- loki
- metallb
- monitoring-kubernetes
- monitoring-ping
- node-manager
- operator-trivy
- pod-reloader
- prometheus
- prometheus-metrics-adapter
- registrypackages
- runtime-audit-engine
- terraform-manager
- user-authn
- vertical-pod-autoscaler
- static-routing-manager

Перезапуск компонентов

После обновления DKP до версии 1.68 будут перезапущены следующие компоненты:

- Kubernetes Control Plane
- Ingress controller
- Prometheus, Grafana
- admission-policy-engine
- chrony
- cloud-provider-azure
- cloud-provider-gcp
- cloud-provider-openstack
- cloud-provider-yandex
- cloud-provider-zvirt
- cni-cilium
- control-plane-manager
- descheduler
- documentation
- extended-monitoring
- ingress-nginx
- istio
- kube-state-metrics
- log-shipper
- loki
- metallb
- monitoring-kubernetes
- monitoring-ping
- node-manager
- openvpn
- operator-trivy
- prometheus
- prometheus-metrics-adapter
- pod-reloader
- registrypackages
- runtime-audit-engine
- service-with-healthchecks
- static-routing-manager
- terraform-manager
- user-authn
- vertical-pod-autoscaler

Полный список изменений, включая перечень обновлённых компонентов, доступен в журнале изменений (changelog) на GitHub.

Встроенные модули Deckhouse Kubernetes Platform

Модуль admission-policy-engine

Модуль admission-policy-engine реализует поддержку admission-политик безопасности в кластере Kubernetes.

Admission-политики — это правила, которые применяются к объектам (например Pod и Service) в момент их создания и изменения в кластере (но не в процессе их работы), на основе информации, представленной в их манифесте. Эти политики направлены на формализацию параметров которые разрешены или запрещены в манифестах объектов.

В DKP политики разделены на три категории:

- Pod Security Standards — политики, реализующие соответствующие Pod Security Standards.
- Операционные политики — политики для создания дополнительных требований к объектам, с помощью валидации значений параметров **не связанных напрямую** с безопасностью (например, список допустимых префиксов для образов контейнеров, политика скачивания образов, список необходимых проб для контейнеров и т.д.).
- Политики безопасности — политики для создания дополнительных требований к объектам, с помощью валидации значений параметров, связанных с безопасностью (например, доступ контейнеров к IPC- или PID-пространству имен хоста, список привилегий для контейнеров и т.д.).

i

Эти политики дополняют друг друга. Если для одного неймспейса применены несколько политик, выполняется валидация объектов по каждой из них. Если хоть одна политика будет нарушена, объект создан не будет.

Особенности отображения сообщений о неудачной валидации объектов

В зависимости от способа создания подов есть особенности формирования сообщений от API о неудачной валидации (нарушении установленных политик):

- Если под создается напрямую, ошибка валидации возвращается в ответе от API о неудачной валидации (нарушении политики).
- Если поды создаются через Deployment, создаётся требуемое количество ReplicaSet, которые, в свою

очередь, пытаются создать поды. В этом случае ошибка валидации не возвращается в ответе API, а отображается в событиях неймспейса или соответствующего ReplicaSet.

Валидация подов при изменении политики или добавлении новой

Для всех трех категорий политик (Pod Security Standards, операционные и политики безопасности) не предусмотрено автоматическое пересоздание существующих подов при изменении действующих или добавлении новых политик. Поды, существовавшие до момента внесения изменений в используемую политику или до добавления новой, продолжат работать до перезапуска. А при перезапуске они будут валидироваться по новым правилам.

В модуле admission-policy-engine для таких случаев предусмотрены алерты (kind: ClusterObservabilityAlert), информирующие о наличии в неймспейсе подов с нарушениями после изменения существующей политики или добавления новой.

Для получения списка алертов используйте команду:

```
d8 k get clusterobservabilityalerts
```

Пример ответа:

NAME	SEVERITY	STATUS	DURATION	SUMMARY
AGE				
SecurityPolicyViolation-f3a77d1dd2175402-1777370195	1	Firing	5h	
Alerting PrometheusUnavailable			5h1m	
OperationPolicyViolation-9b21d0c871796913-1777370435	1	Firing	6h	
Alerting PrometheusUnavailable			6h1m	

Для просмотра информации о конкретном алерте используйте команду:

```
d8 k get clusterobservabilityalert OperationPolicyViolation-9b21d0c871796913-1777370435 -oyaml
```

Пример алерта при нарушении политики Pod Security Standards...


```

kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: PodSecurityStandardsViolation-91e71759e048a397-1777369535
  resourceVersion: "7454828154578800069"
  creationTimestamp: 2026-04-28T09:45:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: PodSecurityStandardsViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: |-
      You have configured [Pod Security Standards]
      (https://kubernetes.io/docs/concepts/security/pod-security-standards/), and one or more
      running pods are violating these standards.

      To identify violating pods:

      - Run the following Prometheus query:

      ```prometheus
 count by (violating_namespace, violating_name, violation_msg) (
 d8_gatekeeper_exporter_constraint_violations{
 violation_enforcement="deny",
 violating_namespace=~".*",
 violating_kind="Pod",
 source_type="PSS"
 }
)
      ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one pod violates the configured cluster pod security standards.
    expr:
      (count(d8_gatekeeper_exporter_constraint_violations{source_type="PSS",violating_kind="Pod",violating_namespace=~".*",violation_enforcement="deny"}))
      > 0
    created_by: observability
    rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T09:45:35Z
  resolvedAt: null
  duration: 20h40m1.015261771s

```

Пример алерта при нарушении операционной политики...

```

kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: OperationPolicyViolation-9b21d0c871796913-1777370435
  resourceVersion: "7454831929456594373"
  creationTimestamp: 2026-04-28T10:00:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: OperationPolicyViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: >-
      You have configured operation policies for the cluster, and one or more
      existing objects are violating these policies.

      To identify violating objects:

      - Run the following Prometheus query:

      ```prometheus
 count by (violating_namespace, violating_kind, violating_name, violation_msg) (
 d8_gatekeeper_exporter_constraint_violations{
 violation_enforcement="deny",
 source_type="OperationPolicy"
 }
)
      ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one object violates the configured cluster operation policies.
  expr:
    (count(d8_gatekeeper_exporter_constraint_violations{source_type="OperationPolicy",violation_
    enforcement="deny"}))
    > 0
  created_by: observability
  rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T10:00:35Z
  resolvedAt: null
  duration: 20h23m41.023025059s

```

Пример алерта при нарушении политики безопасности...

```

kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: SecurityPolicyViolation-f3a77d1dd2175402-1777370195
  resourceVersion: "7454830922622307781"
  creationTimestamp: 2026-04-28T09:56:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: SecurityPolicyViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: >-
      You have configured security policies for the cluster, and one or more
      existing objects are violating these policies.

      To identify violating objects:

      - Run the following Prometheus query:

      ```prometheus
 count by (violating_namespace, violating_kind, violating_name, violation_msg) (
 d8_gatekeeper_exporter_constraint_violations{
 violation_enforcement="deny",
 source_type="SecurityPolicy"
 }
)
      ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one object violates the configured cluster security policies.
  expr:
    (count(d8_gatekeeper_exporter_constraint_violations{source_type="SecurityPolicy",violation_e
    nforcement="deny"}))
    > 0
  created_by: observability
  rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T09:56:35Z
  resolvedAt: null
  duration: 20h29m21.015479019s

```

Pod Security Standards

Pod Security Standards (PSS) — это официальный стандарт Kubernetes, который определяет три уровня безопасности для подов, ограничивая их привилегии. Ограничение происходит с помощью запрета установки определенных параметров в манифесте пода.

Используется многослойная структура — каждый более высокий уровень защиты использует все правила предыдущего уровня и добавляет свои.

В PSS регламентированы следующие уровни защиты (политики):

- **Privileged** — неограничивающая политика с максимально широким уровнем разрешений (отсутствие ограничений).
- **Baseline** — минимально ограничивающая политика, которая предотвращает наиболее известные и популярные способы повышения привилегий. Позволяет использовать стандартную (минимально заданную) конфигурацию пода.
- **Restricted** — политика со значительными ограничениями. Предъявляет самые жёсткие требования

к подам.

i В Deckhouse Kubernetes Platform эти политики реализуются средствами Gatekeeper и контролируются admission-контроллерами модуля `admission-policy-engine`, а не контролером Pod Security Admission от Kubernetes. Из Kubernetes взяты только описания политик.

Подробнее про каждый набор политик и их ограничения можно прочитать в документации Kubernetes.

Политика PSS для неймспейса включается через добавление на него специального лейбла `security.deckhouse.io/pod-policy=<POLICY_NAME>`. Политику по умолчанию можно переопределить глобально (в настройках модуля).

i Модуль не применяет политики к системным неймспейсам.

i При включении модуля `multitenancy-manager` он создаёт свои объекты `OperationPolicy` (например, в неймспейсе `default`). На них не влияют настройки `podSecurityStandards`.

Пример установки политики `Restricted` для всех подов в неймспейсе `my-namespace`:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
```

Дополнительно возможна настройка режима работы политики. Поддерживаются следующие режимы:

- `deny` — запретить запуск подов, не удовлетворяющих политике;
- `warn` — запускать поды, не удовлетворяющие политике, но выдавать предупреждение.
- `dryrun` — запускать поды не удовлетворяющие политике, не выдавать предупреждение пользователю, но фиксировать нарушения в отчетах безопасности;

Настройка режима работы политики производится путем установки лейбла `security.deckhouse.io/pod-policy-action=<POLICY_ACTION>` на соответствующем неймспейсе. Чтобы задать режим работы политик глобально, используйте параметр `enforcementaction`.

Пример установки “warn” режима политик PSS для всех подов в неймспейсе `my-namespace`:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
```

Операционные политики

Операционные политики — это правила, направленные на достижение лучших практик безопасности приложений, но **не относящиеся** напрямую к валидации классических параметров, связанных с безопасностью (например, список допустимых префиксов для образов контейнеров, политика скачивания образов, список необходимых проб для контейнеров и т.д.).

Операционные политики описываются с помощью кастомного ресурса `OperationPolicy`. В нём каждый параметр отвечает за отдельную проверку, применяемую к ресурсам. Использование кастомного ресурса `OperationPolicy` позволяет создавать дополнительные требования к создаваемым ресурсам (высокоуровневые декларативные операционные политики) без явной работы с Gatekeeper.

Рекомендуется устанавливать следующий минимальный набор операционных политик:

```

apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  enforcementAction: Deny
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.ru
  requiredResources:
    limits:
      - memory
    requests:
      - cpu
      - memory
  disallowedImageTags:
    - latest
  requiredProbes:
    - livenessProbe
    - readinessProbe
  maxRevisionHistoryLimit: 3
  imagePullPolicy: Always
  priorityClassNames:
    - production-high
    - production-low
  checkHostNetworkDNSPolicy: true
  checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          custom-operation-policy/enabled: "true"

```

Применение политики реализовано через настройки, расположенные в параметре `spec.match`.

При указании:

```

match:
  namespaceSelector:
    labelSelector:
      matchLabels:
        custom-operation-policy/enabled: "true"

```

Для применения приведённой политики достаточно добавить лейбл `custom-operation-policy/enabled: "true"` на желаемый неймспейс.

В отличие от PSS, название лейбла может быть любым. Требуется лишь совпадение лейбла в селекторе политик и соответствующего неймспейса.

Более подробную информацию об использовании селекторов вы можете прочитать в описании настройки селекторов.

Для политики также возможно указание применяемого действия. Для этого используется параметр `spec.enforcementAction`. Поддерживаются следующие режимы:

- `Deny` — запретить запуск подов не удовлетворяющих политике;
- `Warn` — запускать поды не удовлетворяющие политике, но выдавать предупреждение.
- `Dryrun` — запускать поды не удовлетворяющие политике, не выдавать предупреждение пользователю, но фиксировать нарушения в отчетах безопасности;

Основываясь на этом примере, вы можете создать собственную политику с необходимыми настройками.

Политики безопасности

Политики безопасности — это правила, направленные на достижение лучших практик безопасности приложений с помощью валидации значений параметров, связанных с безопасностью (например, доступ контейнеров к IPC- или PID-пространству имен хоста, список привилегий для контейнеров и т.д.).

Политики безопасности описываются с помощью кастомного ресурса `SecurityPolicy`. В нём каждый параметр отвечает за отдельную проверку применяемую к ресурсам. С помощью этого ресурса возможно сконструировать политику безопасности аналогичную политике PSS любого уровня. Использование кастомного ресурса `SecurityPolicy` позволяет создавать дополнительные требования к создаваемым

ресурсам (высокоуровневые декларативные политики безопасности) без явной работы с Gatekeeper.

Пример политики безопасности:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
      - driver: vmware
    allowedHostPorts:
      - max: 4000
        min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
      - unconfined
    allowedUnsafeSysctls:
      - kernel.*
    allowedVolumes:
      - hostPath
      - projected
    fsGroup:
      ranges:
        - max: 200
          min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
      - ALL
    runAsGroup:
      ranges:
        - max: 500
          min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
        - max: 200
          min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
        - my_profile.json
      allowedProfiles:
        - Localhost
    supplementalGroups:
      ranges:
        - max: 133
          min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy: mypolicy
```



Параметры `allowPrivilegeEscalation` и `allowPrivileged` по умолчанию имеют значение `false` — даже если не указаны явно. Это означает, что контейнеры не смогут запускаться в привилегированном режиме или повышать привилегии. Чтобы разрешить такое поведение, задайте параметр в `true`.

Применение политики реализовано через настройки расположенные в параметре `spec.match`.

При указании:

```
match:
  namespaceSelector:
    labelSelector:
      matchLabels:
        security-policy: mypolicy
```

Для применения приведённой политики достаточно добавить лейбл `security-policy: mypolicy` на желаемый неймспейс.

В отличие от PSS, название лейбла может быть любым. Требуется лишь совпадение лейбла в селекторе политик и соответствующего неймспейса.

Более подробную информацию о использовании селекторов вы можете прочитать в описании настройки селекторов.

Для политики также возможно указание применяемого действия. Для этого используется параметр `spec.enforcementAction`. Поддерживаются следующие режимы:

- `Deny` — запретить запуск подов, не удовлетворяющих политике;
- `Warn` — запускать поды, не удовлетворяющие политике, но выдавать предупреждение.
- `Dryrun` — запускать поды не удовлетворяющие политике, не выдавать предупреждение пользователю, но фиксировать нарушения в отчетах безопасности;

Изменение ресурсов Kubernetes

Модуль позволяет использовать кастомные ресурсы Gatekeeper для модификации объектов в кластере, такие как:

- `AssignMetadata` — для изменения секции `metadata` в ресурсе;
- `Assign` — для изменения других полей, кроме `metadata`;
- `ModifySet` — для добавления или удаления значений из списка, например аргументов для запуска контейнера.
- `AssignImage` — для изменения параметра `image` ресурса.

Подробнее про доступные варианты можно прочитать в документации Gatekeeper.

Настройки

В модуле реализовано 4 алерта.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `admission-policy-engine`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left is a sidebar with navigation links: Обзор, Обновления, Модули (highlighted), Глобальные настройки, and Kubernetes. The main content area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. It contains a table of modules with columns for Name, Status, and Stage. The 'admission-policy-engine' module is shown with a status of 'Включен'. To the right of the table, there is a detailed view for the 'admission-policy-engine' module, showing a 'Включен' status and a toggle switch labeled 'Модуль включен' which is currently turned on. Below this, there are links for 'Конфигурация', 'Ресурсы', and 'YAML', and a link to 'Дополнительные настройки'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable`

для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `admission-policy-engine`:

```
d8 system module enable admission-policy-engine
```

- **С помощью ModuleConfig `admission-policy-engine`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `admission-policy-engine` (создайте его, при необходимости).

Пример манифеста для включения модуля `admission-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  enabled: true
```

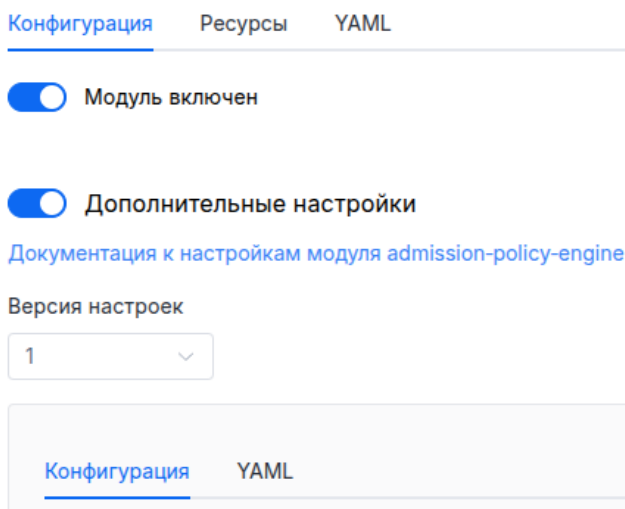
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `admission-policy-engine`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `admission-policy-engine` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `admission-policy-engine`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `admission-policy-engine` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc admission-policy-engine
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `admission-policy-engine`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```


Пример файла манифеста ModuleConfig `admission-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.denyVulnerableImages**

объект

Доступно в редакциях: EE, CSE Lite (1.73), CSE Pro (1.73)

Настройки Trivy-провайдера.

Trivy-провайдер запрещает создание ресурсов Pod, Deployment, StatefulSet и DaemonSet с образами, содержащими уязвимости, в неймспейсах с лейблом `security.deckhouse.io/trivy-provider: ""`.

Устарело. Эти настройки были перенесены в модуль `operator-trivy`. Пожалуйста, мигрируйте вашу конфигурацию.

- **settings.denyVulnerableImages.allowedSeverityLevels**

массив строк

Список допустимых уровней критичности уязвимости.

Не будут запрещены образы контейнеров, содержащие только уязвимости указанных уровней.

- **Элемент массива**

строка

Допустимые значения: `UNKNOWN`, `LOW`, `MEDIUM`, `HIGH`, `CRITICAL`

- **settings.denyVulnerableImages.enabled**

булевый

Включить Trivy-провайдер.

По умолчанию: `false`

- **settings.denyVulnerableImages.registrySecrets**

массив объектов

Список дополнительных секретов для доступа к приватным хранилищам образов контейнеров.

По умолчанию для загрузки образов для сканирования используется секрет `deckhouse-registry`.

По умолчанию: `[]`

- **settings.denyVulnerableImages.registrySecrets.name**

строка

Обязательный параметр

- **settings.denyVulnerableImages.registrySecrets.namespace**

строка

Обязательный параметр

- **settings.denyVulnerableImages.storageClass**

строка

Имя StorageClass, который используется Trivy-провайдером.

Если значение не задано, будет использоваться StorageClass, заданный в глобальном параметре `settings.modules.storageClass`.

Глобальный параметр `settings.modules.storageClass` применяется только при

включении модуля. Изменение глобального параметра при включенном модуле не приведет к перезаказу диска.

i **Внимание.** Если указать значение, отличное от текущего StorageClass, используемого существующим PersistentVolumeClaim, диск будет перезаказан, и все данные удалятся.

Если указать `false`, вместо PVC будет принудительно использоваться `emptyDir`.

Примеры:

```
storageClass: ceph-ssd
```

```
storageClass: 'false'
```

◦ **settings.podSecurityStandards** объект

Настройки политик Pod Security Standards (PSS).

■ **settings.podSecurityStandards.defaultPolicy** строка

Определяет политику Pod Security Standards по умолчанию для всех **несистемных** пространств имен:

- `Privileged` — политика без ограничений. Данная политика допускает эскалацию привилегий;
- `Baseline` — политика с минимальными ограничениями, ограничивающая использование эскалаций привилегий;
- `Restricted` — политика с максимальными ограничениями, соответствующая актуальным рекомендациям по безопасному запуску приложений в кластере.

По умолчанию:

- `Baseline` — при первичной установке Deckhouse версии v1.55 и выше;
- `Privileged` — при первичной установке Deckhouse версии ниже v1.55 (обновление Deckhouse в кластере на версию v1.55 и выше **не меняет** политику по умолчанию на `Baseline`).

Допустимые значения: `Privileged`, `Baseline`, `Restricted`

■ **settings.podSecurityStandards.enforcementAction** строка

Действие, которое будет выполнено по результатам проверки ограничений:

- `Deny` — запрет;
- `Dryrun` — отсутствие действия. Применяется при отладке. Информацию о событии можно посмотреть в Grafana или консоли с помощью `d8 k`;
- `Warn` — аналогично `Dryrun`, но дополнительно к информации о событии будет выведена информация о том, из-за какого ограничения (constraint) был бы запрет действия, если бы вместо `Warn` использовался `Deny`.

По умолчанию: `Deny`

Допустимые значения: `Warn`, `Deny`, `Dryrun`

■ **settings.podSecurityStandards.policies** объект

Определяет дополнительные параметры политик

■ **settings.podSecurityStandards.policies.hostPorts** объект

Настройки ограничения HostPort.

■ **settings.podSecurityStandards.policies.hostPorts.knownRanges** массив объектов

Список диапазонов портов, которые будут разрешены в привязке hostPort.

- **settings.podSecurityStandards.policies.hostPorts.knownRanges.max**
целочисленный
- **settings.podSecurityStandards.policies.hostPorts.knownRanges.min**
целочисленный



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

OperationPolicy

Scope: Cluster

Version: v1alpha1

Описывает операционную политику для кластера.

Каждый ресурс `OperationPolicy` описывает правила для объектов в кластере.

- **spec**

объект

Обязательный параметр

- **spec.enforcementAction**

строка

Действие, которое будет выполнено по результатам проверки ограничений:

- `Deny` — запрет;
- `Dryrun` — отсутствие действия. Применяется при отладке. Информацию о событии можно посмотреть в Grafana в Dashboard Security/Admission policy engine;
- `Warn` — аналогично `Dryrun`, но дополнительно к информации о событии будет выведена информация о том, из-за какого ограничения (constraint) был бы запрет действия, если бы вместо `Warn` использовался `Deny`.

По умолчанию: `Deny`

Допустимые значения: `Warn`, `Deny`, `Dryrun`

- **spec.match**

объект

Обязательный параметр

- **spec.match.labelSelector**

объект

Указывает селектор лейблов для фильтрации подов.

Больше информации в документации.

- **spec.match.labelSelector.matchExpressions**

массив объектов

Список выражений лейблов для подов.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.match.labelSelector.matchExpressions.key**

строка

Обязательный параметр

- **spec.match.labelSelector.matchExpressions.operator**

строка

Обязательный параметр

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.labelSelector.matchExpressions.values**

массив строк

- **spec.match.labelSelector.matchLabels**

объект

Список лейблов, которые должен иметь под.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.match.namespaceSelector**

объект

Указывает селектор пространства имен для фильтрации объектов.

- **spec.match.namespaceSelector.excludeNames**

массив строк

Включить все пространства имен, кроме определенного набора.

- **spec.match.namespaceSelector.labelSelector**

объект

Указывает селектор лейблов для фильтрации пространств имен.

Больше информации в документации.

- **spec.match.namespaceSelector.labelSelector.matchExpressions**

массив объектов

Список выражений лейблов для пространств имен.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.match.namespaceSelector.labelSelector.matchExpressions.key**

строка

Обязательный параметр

- **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**

строка

Обязательный параметр

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.namespaceSelector.labelSelector.matchExpressions.values**

массив строк

- **spec.match.namespaceSelector.labelSelector.matchLabels**

объект

Список лейблов, которые должно иметь пространство имен.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.match.namespaceSelector.matchNames**

массив строк

Включать только определенный набор пространств имен.

- **spec.policies**

объект

Обязательный параметр

- **spec.policies.allowedRepos**

массив строк

Список префиксов, которые может иметь образ контейнера.

- **Элемент массива**

строка

Пример:

```
registry.deckhouse.io
```

- **spec.policies.checkContainerDuplicates**

булевый

Проверка имен контейнеров и переменных env на наличие дубликатов.

- **spec.policies.checkHostNetworkDNSPolicy**

булевый

Проверка, что dnsPolicy ClusterFirstWithHostNet установлена для подов с hostNetwork: true.

- **spec.policies.disallowedImageTags**

массив строк

Требует, чтобы образы контейнера имели тег, отличный от указанных в списке.

Пример:

```
disallowedImageTags: latest
```

- **spec.policies.disallowedTolerations**

массив объектов

Запрещает создание подов с tolerations из указанного списка.

Больше информации в документации Kubernetes.

Список шаблонов toleration (partial match по указанным полям: key/operator/value/effect).

- **spec.policies.disallowedTolerations.effect**

строка

Допустимые значения: NoSchedule, PreferNoSchedule, NoExecute

- **spec.policies.disallowedTolerations.key**

строка

- **spec.policies.disallowedTolerations.operator**

строка

Допустимые значения: Exists, Equal

- **spec.policies.disallowedTolerations.value**

строка

- **spec.policies.imagePullPolicy**

строка

Требуемая политика скачивания образов для контейнеров.

Допустимые значения: Always, IfNotPresent

- **spec.policies.ingressClassNames**

массив строк

Список возможных классов ингресса.

- **spec.policies.maxRevisionHistoryLimit**

целочисленный

Максимальное значение для истории ревизий.

- **spec.policies.priorityClassNames**

массив строк

Список возможных классов приоритета.

- **spec.policies.replicaLimits**

объект

Проверка диапазона разрешенных реплик. Значения включаются в диапазон.

- **spec.policies.replicaLimits.maxReplicas**

целочисленный

Максимально разрешенное количество реплик, включительно.

- **spec.policies.replicaLimits.minReplicas**

целочисленный

Минимально разрешенное количество реплик, включительно.

- **spec.policies.requiredAnnotations**

объект

Список аннотаций, которые должен указать объект.

- **spec.policies.requiredAnnotations.annotations**

массив объектов

- **spec.policies.requiredAnnotations.annotations.allowedRegex**

строка

Если указано, то содержит регулярное выражение, которому должно соответствовать значение аннотации. Значение должно содержать хотя бы одно совпадение с регулярным выражением.

- **spec.policies.requiredAnnotations.annotations.key**

строка

Требуемая аннотация.

- **spec.policies.requiredAnnotations.watchKinds**

массив строк

Список объектов Kubernetes в формате `$apiGroup/$kind` для просмотра аннотаций.

- **Элемент массива**

строка

Шаблон: `^[a-z]*/[a-zA-Z]+$`

Примеры:

```
apps/Deployment
```

```
"/Pod"
```

```
networking.k8s.io/Ingress
```

- **spec.policies.requiredLabels**

объект

Список лейблов, которые должен указать объект.

- **spec.policies.requiredLabels.labels**

массив объектов

- **spec.policies.requiredLabels.labels.allowedRegex**

строка

Если указано, то содержит регулярное выражение, которому должно соответствовать значение лейбла. Значение должно содержать хотя бы одно совпадение с регулярным выражением.

- **spec.policies.requiredLabels.labels.key**

строка

Требуемый лейбл.

- **spec.policies.requiredLabels.watchKinds**

массив строк

Список объектов Kubernetes в формате `$apiGroup/$kind` для просмотра лейблов.

- **Элемент массива**

строка

Шаблон: `^[a-z]*/[a-zA-Z]+$`

Примеры:

```
apps/Deployment
```

```
"/Pod"
```

```
networking.k8s.io/Ingress
```

- **spec.policies.requiredProbes**

массив строк

Список проб, которые необходимы (например, `readinessProbe` и `livenessProbe`).

Пример:

```
requiredProbes:
- readinessProbe
- livenessProbe
```

- **Элемент массива**
строка

Допустимые значения: `livenessProbe`, `readinessProbe`, `startupProbe`

- **spec.policies.requiredResources**
объект

Набор ресурсов, который требуют контейнеры.

- **spec.policies.requiredResources.limits**
массив строк

Список ограничений по выделяемым ресурсам, которые должны соблюдаться (по процессору, памяти или оба ограничения).

- **Элемент массива**
строка

Допустимые значения: `cpu`, `memory`

- **spec.policies.requiredResources.requests**
массив строк

Список запросов на выделение ресурсов, которые должны быть выполнены (процессор, память или оба запроса).

- **Элемент массива**
строка

Допустимые значения: `cpu`, `memory`

- **spec.policies.storageClassNames**
массив строк

Список возможных классов хранилища.

SecurityPolicyException

Scope: Namespaced

Version: v1alpha1

Описывает исключения из политики безопасности кластера. Каждый ресурс SecurityPolicyException задает перечень конкретных разрешений.

- **status**
объект
 - **status.deckhouse**
объект

- **status.deckhouse.observed**
объект

Contains last timestamp when the resource change was noted by the operator and its checksum.

- **status.deckhouse.observed.checkSum**
строка

The checksum of last observed resource.

- **status.deckhouse.observed.lastTimestamp**
строка

A timestamp of when the operator last noted a change to the resource.

- **status.deckhouse.processed**
объект

Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

- **status.deckhouse.processed.checkSum**
строка

The checksum of last applied resource.

- **status.deckhouse.processed.lastTimestamp**
строка

A timestamp of when the resource was last applied in the cluster.

- **status.deckhouse.synced**
строка

True if last observed version of the resource was successfully applied in the cluster.

- **spec**
объект

Обязательный параметр

- **spec.network**
объект

Исключения для сетевых параметров.

- **spec.network.hostIPC**
объект

Исключения для параметра `hostIPC`.

- **spec.network.hostIPC.allowedValue**
булевый

Обязательный параметр

Допустимое значение для `hostIPC`.

По умолчанию: `false`

- **spec.network.hostIPC.metadata**
объект

Метаданные, связанные с исключением.

- **spec.network.hostNetwork**
объект

Исключения для параметра `hostNetwork`.

- **spec.network.hostNetwork.allowedValue**
булевый

Обязательный параметр

Допустимое значение для `hostNetwork`.

По умолчанию: `false`

- **spec.network.hostNetwork.metadata**
объект

Метаданные, связанные с исключением.

- **spec.network.hostPID**
объект

Исключения для параметра `hostPID`.

- **spec.network.hostPID.allowedValue**
булевый

Обязательный параметр

Допустимое значение для `hostPID`.

По умолчанию: `false`

- **spec.network.hostPID.metadata**
объект

Метаданные, связанные с исключением.

- **spec.network.hostPorts**
массив объектов

Исключения для хостовых портов. Для подов с `hostNetwork: true` исключения распространяются на порты, указанные в `containerPort`.

- **spec.network.hostPorts.metadata**
объект

Метаданные, связанные с исключением.

- **spec.network.hostPorts.port**
целочисленный

Разрешенный хостовой порт.

- **spec.network.hostPorts.protocol**
строка

Протокол для хостового порта.

Допустимые значения: TCP, UDP

- **spec.securityContext**
объект

Исключения для параметров секции securityContext.

- **spec.securityContext.allowPrivilegeEscalation**
объект

Исключения для параметра allowPrivilegeEscalation.

- **spec.securityContext.allowPrivilegeEscalation.allowedValue**
булевый

Обязательный параметр

Допустимое значение для allowPrivilegeEscalation.

По умолчанию: false

- **spec.securityContext.allowPrivilegeEscalation.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.appArmorProfile**
объект

Список разрешенных значений профилей AppArmor для подов и контейнеров.

Поддерживаемые форматы:

- через аннотации: runtime/default, docker/default, unconfined, localhost/some-profile.json. localhost/* разрешает любой локальный профиль;
- через securityContext: RuntimeDefault, Unconfined, Localhost. Для Localhost укажите разрешенные профили с помощью параметра allowedLocalhostFiles.

Типы профилей:

- Unconfined — без ограничений (не рекомендуется по соображениям безопасности);
- RuntimeDefault — профиль по умолчанию, предоставляемый container runtime (например, Docker, CRI-O);
- Localhost — пользовательский профиль, определенный на хосте и настроенный под приложение.

Использование * разрешает все профили. Необязательно указывать оба формата — они автоматически накладываются друг на друга.

- **spec.securityContext.appArmorProfile.allowedValues**
массив строк

Обязательный параметр

Разрешенные значения:

- runtime/default;
- unconfined;
- localhost/* (где * может быть именем профиля или подстановкой (wildcard)).
- **Элемент массива**
строка

Шаблон: ^(runtime/default|unconfined|localhost/.+)\$

- **spec.securityContext.appArmorProfile.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.capabilities**
объект

Исключения для параметра capabilities.

- **spec.securityContext.capabilities.allowedValues**
объект

Обязательный параметр

Разрешенные значения для добавления или удаления Linux-привилегий (capabilities).

- **spec.securityContext.capabilities.allowedValues.add**
массив строк

Список привилегий, которые разрешено добавлять контейнеру.

По умолчанию: `[]`

- **spec.securityContext.capabilities.allowedValues.drop**
массив строк

Список привилегий, которые разрешено удалять у контейнера.

По умолчанию: `["all"]`

- **spec.securityContext.capabilities.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.privileged**
объект

Исключения для параметра `privileged`.

- **spec.securityContext.privileged.allowedValue**
булевый

Обязательный параметр

Допустимое значение для `privileged`.

По умолчанию: `false`

- **spec.securityContext.privileged.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.procMount**
объект

Исключения для параметра `procMount`.

- **spec.securityContext.procMount.allowedValues**
массив строк

Обязательный параметр

Список разрешенных значений `procMount`.

- **Элемент массива**
строка

Допустимые значения: `Default`, `Unmasked`

- **spec.securityContext.procMount.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.readOnlyRootFilesystem**
объект

Исключения для параметра `readOnlyRootFilesystem`.

- **spec.securityContext.readOnlyRootFilesystem.allowedValue**
булевый

Обязательный параметр

Допустимое значение для `readOnlyRootFilesystem`.

По умолчанию: `false`

- **spec.securityContext.readOnlyRootFilesystem.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.runAsNonRoot**
объект

Исключения для параметра `runAsNonRoot`.

- **spec.securityContext.runAsNonRoot.allowedValue**

булевый

Обязательный параметр

Допустимое значение для `runAsNonRoot`.

По умолчанию: `true`

- **spec.securityContext.runAsNonRoot.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.runAsUser**
объект

Исключения для параметра `runAsUser`.

- **spec.securityContext.runAsUser.allowedValues**
массив целых чисел

Обязательный параметр

Список разрешенных идентификаторов пользователей (UID).

- **spec.securityContext.runAsUser.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.seLinuxOptions**
объект

Исключения для параметра `seLinuxOptions`.

- **spec.securityContext.seLinuxOptions.allowedValues**
массив объектов

Обязательный параметр

Список разрешенных конфигураций SELinux.

- **spec.securityContext.seLinuxOptions.allowedValues.level**
строка

Лейбл уровня SELinux.

- **spec.securityContext.seLinuxOptions.allowedValues.role**
строка

Лейбл роли SELinux.

- **spec.securityContext.seLinuxOptions.allowedValues.type**
строка

Лейбл типа SELinux.

- **spec.securityContext.seLinuxOptions.allowedValues.user**
строка

Лейбл пользователя SELinux.

- **spec.securityContext.seLinuxOptions.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.seccompProfile**
объект

Исключения для параметра `seccompProfile`.

- **spec.securityContext.seccompProfile.allowedValues**
массив строк

Обязательный параметр

Список разрешенных seccomp-профилей.

- **spec.securityContext.seccompProfile.metadata**
объект

Метаданные, связанные с исключением.

- **spec.securityContext.sysctls**
объект

Исключения для параметра `sysctl`.

- **spec.securityContext.sysctl.allowedValues**

массив объектов

Обязательный параметр

Список разрешенных настроек `sysctl`.

- **spec.securityContext.sysctl.allowedValues.name**

строка

Имя параметра `sysctl`.

- **spec.securityContext.sysctl.allowedValues.value**

строка

Допустимое значение для указанного параметра `sysctl`.

- **spec.securityContext.sysctl.metadata**

объект

Метаданные, связанные с исключением.

- **spec.volumes**

объект

Исключения для параметров томов.

- **spec.volumes.hostPath**

объект

Исключения для `hostPath`-томов.

- **spec.volumes.hostPath.allowedValues**

массив объектов

Обязательный параметр

Список разрешенных конфигураций `hostPath`.

- **spec.volumes.hostPath.allowedValues.metadata**

объект

Метаданные, связанные с исключением.

- **spec.volumes.hostPath.allowedValues.metadata.description**

строка

Описание исключения.

- **spec.volumes.hostPath.allowedValues.path**

строка

Разрешенный путь `hostPath`.

- **spec.volumes.hostPath.allowedValues.readOnly**

булевый

Значение `readOnly` для `hostPath`.

По умолчанию: `false`

- **spec.volumes.types**

объект

Исключения по типам томов.

- **spec.volumes.types.allowedValues**

массив строк

Обязательный параметр

Список разрешенных типов томов.

- **spec.volumes.types.metadata**

объект

Метаданные, связанные с исключением.

SecurityPolicy

Scope: Cluster
Version: v1alpha1

Описывает политику безопасности для кластера.

Каждый custom resource `SecurityPolicy` задает правила для объектов в кластере.

- **status**

объект

- **status.deckhouse**

объект

- **status.deckhouse.observed**

объект

Contains last timestamp when the resource change was noted by the operator and its checksum.

- **status.deckhouse.observed.checkSum**

строка

The checksum of last observed resource.

- **status.deckhouse.observed.lastTimestamp**

строка

A timestamp of when the operator last noted a change to the resource.

- **status.deckhouse.processed**

объект

Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

- **status.deckhouse.processed.checkSum**

строка

The checksum of last applied resource.

- **status.deckhouse.processed.lastTimestamp**

строка

A timestamp of when the resource was last applied in the cluster.

- **status.deckhouse.synced**

строка

True if last observed version of the resource was successfully applied in the cluster.

- **spec**

объект

Обязательный параметр

- **spec.enforcementAction**

строка

Реакция на нарушение политики:

- `Deny` — запрет;
- `Dryrun` — событие без действия. Применяется при отладке. Информацию о событии можно посмотреть в Grafana в Dashboard `Security/Admission policy engine`;
- `Warn` — событие без действия, аналогичное `Dryrun`. Содержит информацию об ограничении (constraint), которое вызвало бы запрет, если бы было использовано действие `Deny`.

По умолчанию: `Deny`

Допустимые значения: `Warn`, `Deny`, `Dryrun`

- **spec.match**

объект

Обязательный параметр

Условия фильтрации контейнеров. С помощью условий можно выбрать группу подов или контейнеров, к которой будут применены политики.

- **spec.match.labelSelector**

объект

Указывает селектор лейблов для фильтрации подов.

Подробнее в документации Kubernetes.

- **spec.match.labelSelector.matchExpressions**

массив объектов

Список условий лейблов для подов.

Пример:

```
matchExpressions:
```

```
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.match.labelSelector.matchExpressions.key**

строка

Обязательный параметр

- **spec.match.labelSelector.matchExpressions.operator**

строка

Обязательный параметр

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.labelSelector.matchExpressions.values**

массив строк

- **spec.match.labelSelector.matchLabels**

объект

Список лейблов, которые должен иметь под.

Пример:

```
matchLabels:
```

```
foo: bar
baz: who
```

- **spec.match.namespaceSelector**

объект

Указывает селектор пространства имен для фильтрации объектов.

- **spec.match.namespaceSelector.excludeNames**

массив строк

Выбирает все пространства имен, которых нет в списке. Допускается использование шаблонов (glob patterns).

- **spec.match.namespaceSelector.labelSelector**

объект

Указывает селектор лейблов для фильтрации пространств имен.

Подробнее в документации Kubernetes.

- **spec.match.namespaceSelector.labelSelector.matchExpressions**

массив объектов

Список условий лейблов для пространств имен.

Пример:

```
matchExpressions:
```

```
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.match.namespaceSelector.labelSelector.matchExpressions.key**

строка

Обязательный параметр

- **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**

строка

Обязательный параметр

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.namespaceSelector.labelSelector.matchExpressions.values**

массив строк

- **spec.match.namespaceSelector.labelSelector.matchLabels**

объект

Список лейблов, которые должно иметь пространство имен.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.match.namespaceSelector.matchNames**

массив строк

Выбирает пространства имен, соответствующие списку. Допускается использование шаблонов (glob patterns).

- **spec.policies**

объект

Обязательный параметр

Набор политик, условиям которых должны соответствовать поды и контейнеры.

- **spec.policies.allowHostIPC**

булевый

Разрешает контейнерам доступ к IPC-пространству имен хоста.

- **spec.policies.allowHostNetwork**

булевый

Разрешает контейнерам доступ к Network-пространству имен хоста.

- **spec.policies.allowHostPID**

булевый

Разрешает контейнерам доступ к PID-пространству имен хоста.

- **spec.policies.allowPrivilegeEscalation**

булевый

Разрешает процессам в контейнерах запрашивать более широкий список привилегий, чем у родительского процесса.

По умолчанию (если значение не указано) — `false`. Это означает, что при отсутствии данного поля или его явного указания, контейнеру будет отказано в повышении привилегий. Если вы хотите разрешить повышение привилегий, задайте для данного параметра значение `true`.

- **spec.policies.allowPrivileged**

булевый

Разрешает запускать контейнеры в привилегированном режиме.

По умолчанию (если значение не указано) — `false`. Это означает, что при отсутствии данного поля или его явного указания, контейнеру будет отказано в повышении привилегий. Если вы хотите разрешить повышение привилегий, задайте для данного параметра значение `true`.

- **spec.policies.allowRbacWildcards**

булевый

Разрешает использование wildcard (*) в ресурсах Role и RoleBinding (namespace scope).

Ресурсы ClusterRole и ClusterRoleBinding политикой SecurityPolicy не валидируются.

По умолчанию (если значение не указано) — `true`. Это означает, что при отсутствии данного поля или его явного указания, использование wildcard разрешено. Если вы хотите запретить использование wildcard, задайте для данного параметра значение `false`.

По умолчанию: `true`

- **spec.policies.allowedAppArmor**

массив строк

Список разрешенных для контейнеров профилей AppArmor.

Пример:

```
allowedAppArmor:
- runtime/default
- unconfined
```

- **Элемент массива**

строка

Профиль AppArmor.

- **spec.policies.allowedCapabilities**

массив строк

Список привилегий (linux capabilities), доступных контейнерам.

Используйте `ALL`, чтобы разрешить все привилегии.

Пример:

```
allowedCapabilities:
- SETGID
- SETUID
- NET_BIND_SERVICE
```

- **Элемент массива**

строка

Привилегия (linux capability).

Допустимые значения: `ALL`, `SETPCAP`, `SYS_MODULE`, `SYS_RAWIO`, `SYS_PACCT`, `SYS_ADMIN`, `SYS_NICE`, `SYS_RESOURCE`, `SYS_TIME`, `SYS_TTY_CONFIG`, `MKNOD`, `AUDIT_WRITE`, `AUDIT_CONTROL`, `MAC_OVERRIDE`, `MAC_ADMIN`, `NET_ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC_OVERRIDE`, `FOWNER`, `DAC_READ_SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX_IMMUTABLE`, `NET_BIND_SERVICE`, `NET_BROADCAST`, `IPC_LOCK`, `IPC_OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.allowedClusterRoles**

массив строк

Список разрешенных ролей кластера для связывания с пользователями.

- **spec.policies.allowedFlexVolumes**

массив объектов

Список разрешенных драйверов Flex Volumes.

- **spec.policies.allowedFlexVolumes.driver**

строка

Название драйвера.

- **spec.policies.allowedHostPaths**

массив объектов

Список разрешённых префиксов путей для монтирования `hostPath`. Если указан пустой список (`[]`), использование `hostPath` запрещено для всех путей. Если поле отсутствует, ограничения не применяются.

Пример:

```
allowedHostPaths:
- pathPrefix: "/dev"
  readOnly: true
```

- **spec.policies.allowedHostPaths.pathPrefix**

строка

Обязательный параметр

Префикс определяет, какие директории хоста доступны для монтирования в контейнерах.

Использование маски `*` недопустимо. Завершающие слэши отбрасываются при сопоставлении пути и префикса.

Например, префикс `/foo` разрешает пути `/foo`, `/foo/` и `/foo/bar`. Напротив, пути `/food` или `/etc/foo` будут недоступны.

- **spec.policies.allowedHostPaths.readOnly**

булевый

Проверяет, что все `volumeMount`, попадающие в префикс (с параметром `pathPrefix`), монтируются только на чтение (`readOnly: true`).

По умолчанию: `false`

- **spec.policies.allowedHostPorts**

массив объектов

Список диапазонов портов хоста (`hostPort`), разрешенных для использования в контейнерах. Для подов с `hostNetwork: true` происходит дополнительная проверка портов, указанных в `containerPort`.

- **spec.policies.allowedHostPorts.max**
целочисленный

Максимальное значение диапазона.

- **spec.policies.allowedHostPorts.min**
целочисленный

Минимальное значение диапазона.

- **spec.policies.allowedProcMount**
строка

Разрешает монтирование `/proc` для контейнеров.

Допустимые значения: `Default`, `Unmasked`

Пример:

```
allowedProcMount: Unmasked.
```

- **spec.policies.allowedServiceTypes**
массив строк

Список разрешенных типов сервисов.

Пример:

```
allowedServiceTypes:
- LoadBalancer
- ClusterIP
```

- **Элемент массива**
строка

Допустимые значения: `ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`

- **spec.policies.allowedUnsafeSysctls**
массив строк

Список разрешенных небезопасных ключей `sysctl`.

Используйте `*`, чтобы разрешить все ключи.

Пример:

```
allowedUnsafeSysctls:
- kernel.msg*
- net.core.somaxconn
```

- **spec.policies.allowedVolumes**
массив строк

Список разрешенных `volume`-плагинов.

Пример:

```
allowedVolumes:
- hostPath
- persistentVolumeClaim
```

- **Элемент массива**
строка

Допустимые значения: `*`, `none`, `awsElasticBlockStore`, `azureDisk`, `azureFile`, `cephFS`, `cinder`, `configMap`, `csi`, `downwardAPI`, `emptyDir`, `ephemeral`, `fc`, `flexVolume`, `flocker`, `gcePersistentDisk`, `gitRepo`, `glusterfs`, `hostPath`, `iscsi`, `nfs`, `persistentVolumeClaim`, `photonPersistentDisk`, `portworxVolume`, `projected`, `quobyte`, `rbd`, `scaleIO`, `secret`, `storageos`, `vsphereVolume`

- **spec.policies.automountServiceAccountToken**
булевый

Разрешает запускать поды с автоматическим монтированием токена сервисного аккаунта.

- **spec.policies.blockWildcardDomains**
булевый

Запрещает создание объектов Ingress с `*` в DNS доменах.

- **spec.policies.forbiddenSysctl**
массив строк

Список запрещенных ключей sysctl.

Имеет преимущество перед параметром `allowedUnsafeSysctls`.

Пример:

```
forbiddenSysctls:
- kernel.msg*
- net.core.somaxconn
```

- **spec.policies.fsGroup**
объект

Задаёт группы, разрешенные для использования как `fsGroup` в параметре `securityContext`.

- **spec.policies.fsGroup.ranges**
массив объектов

Список диапазонов идентификаторов для `fsGroup`, разрешенных в режиме `MustRunAs`.

- **spec.policies.fsGroup.ranges.max**
целочисленный

Максимальное значение диапазона.

- **spec.policies.fsGroup.ranges.min**
целочисленный

Минимальное значение диапазона.

- **spec.policies.fsGroup.rule**
строка

Обязательный параметр

Правило определения разрешенных значений `fsGroup`.

Допустимые значения: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.readOnlyRootFilesystem**
булевый

Если установлен в `true`, разрешает запуск только тех подов, у всех контейнеров которых корневая файловая система доступна только для чтения (в поле `securityContext` установлено `readOnlyRootFilesystem: true`). Подробнее в документации Kubernetes.

- **spec.policies.requiredDropCapabilities**
массив строк

Список привилегий (linux capabilities), обязательных к исключению.

Используйте `ALL`, чтобы исключить все привилегии.

Пример:

```
requiredDropCapabilities:
- SETGID
- SETUID
- NET_BIND_SERVICE
```

- **Элемент массива**
строка

Привилегия (linux capability).

Допустимые значения: `ALL`, `SETPCAP`, `SYS_MODULE`, `SYS_RAWIO`, `SYS_PACCT`, `SYS_ADMIN`, `SYS_NICE`, `SYS_RESOURCE`, `SYS_TIME`, `SYS_TTY_CONFIG`, `MKNOD`, `AUDIT_WRITE`, `AUDIT_CONTROL`, `MAC_OVERRIDE`, `MAC_ADMIN`, `NET_ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC_OVERRIDE`, `FOWNER`, `DAC_READ_SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX_IMMUTABLE`, `NET_BIND_SERVICE`, `NET_BROADCAST`, `IPC_LOCK`, `IPC_OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.runAsGroup**
объект

Задаёт основные группы (`runAsGroup`), разрешенные для использования в параметре `securityContext`.

- **spec.policies.runAsGroup.ranges**
массив объектов

Список диапазонов идентификаторов групп для `runAsGroup`, разрешенных в режиме `MustRunAs`.

- **spec.policies.runAsGroup.ranges.max**
целочисленный

Максимальное значение диапазона.

- **spec.policies.runAsGroup.ranges.min**
целочисленный

Минимальное значение диапазона.

- **spec.policies.runAsGroup.rule**
строка

Обязательный параметр

Правило определения разрешенных значений `runAsGroup`.

Допустимые значения: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.runAsUser**
объект

Задаёт разрешенные в параметре `securityContext` идентификаторы пользователей (`runAsUser`).

- **spec.policies.runAsUser.ranges**
массив объектов

Список диапазонов идентификаторов пользователей для `runAsUser`, разрешенных в режиме `MustRunAs`.

- **spec.policies.runAsUser.ranges.max**
целочисленный

Максимальное значение диапазона.

- **spec.policies.runAsUser.ranges.min**
целочисленный

Минимальное значение диапазона.

- **spec.policies.runAsUser.rule**
строка

Обязательный параметр

Правило определения разрешенных значений `runAsUser`.

Допустимые значения: `MustRunAs`, `MustRunAsNonRoot`, `RunAsAny`

- **spec.policies.seLinux**
массив объектов

Определяет разрешенные для использования в параметре `securityContext` значения SELinux-лейблов.

- **spec.policies.seLinux.level**
строка

Значения для SELinux level-лейблов.

- **spec.policies.seLinux.role**
строка

Значения для SELinux role-лейблов.

- **spec.policies.seLinux.type**
строка

Значения для SELinux type-лейблов.

- **spec.policies.seLinux.user**
строка

Значения для SELinux user-лейблов.

- **spec.policies.seccompProfiles**
объект

Задаёт список разрешённых seccomp-профилей для аннотаций подов и контейнеров.

- **spec.policies.seccompProfiles.allowedLocalhostFiles**
массив строк

Определяет локальные профили seccomp (в формате JSON), разрешённые после указания `Localhost` в параметре `allowedProfiles`.

Пустой список запрещает использование локальных профилей.

- **spec.policies.seccompProfiles.allowedProfiles**
массив строк

Список разрешённых seccomp-профилей для подов и контейнеров.

Поддерживаются оба формата:

- Через аннотации: `runtime/default`, `docker/default`, `unconfined`, `localhost/some-profile.json`. `localhost/*` — разрешает любые локальные профили.
- Через `securityContext`: `RuntimeDefault`, `Unconfined`, `Localhost`. Для `Localhost` укажите разрешённые профили в параметре `allowedLocalhostFiles`.

Типы профилей:

- `Unconfined` — без ограничений (небезопасно).
- `RuntimeDefault` — стандартный профиль среды выполнения (Docker, CRI-O).
- `Localhost` — пользовательский профиль на узле (гибко и точно под приложение).

Использование `*` разрешает все профили. Не обязательно указывать оба формата — они автоматически сопоставляются.

- **spec.policies.supplementalGroups**
объект

Задаёт дополнительные группы (`supplementalGroups`), разрешённые для использования в параметре `securityContext`.

- **spec.policies.supplementalGroups.ranges**
массив объектов

Список диапазонов идентификаторов групп для `supplementalGroups`, разрешённых в режиме `MustRunAs`.

- **spec.policies.supplementalGroups.ranges.max**
целочисленный

Максимальное значение диапазона.

- **spec.policies.supplementalGroups.ranges.min**
целочисленный

Минимальное значение диапазона.

- **spec.policies.supplementalGroups.rule**
строка

Обязательный параметр

Правило определения разрешённых значений `supplementalGroups`.

Допустимые значения: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.verifyImageSignatures**
массив объектов

Доступно в редакциях: SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Список политик проверки подписи образов контейнеров.

Образы контейнеров должны быть подписаны с помощью инструмента Cosign.

Пример:

```
verifyImageSignatures:
- reference: docker.io/myrepo/*
  publicKeys:
  - |-
    -----BEGIN PUBLIC KEY-----
    .....
    -----END PUBLIC KEY-----
- reference: company.registry.com/*
  dockerCfg: "<Base64_dockerCfg>"
  publicKeys:
  - |-
    -----BEGIN PUBLIC KEY-----
    .....
    -----END PUBLIC KEY-----
```

■ **spec.policies.verifyImageSignatures.ca**

строка

Пользовательский `certificate authority` для подключения к хранилищу образов контейнеров.

■ **spec.policies.verifyImageSignatures.dockerCfg**

строка

Строка в формате Base64 с данными аутентификации в хранилище образов контейнеров.

Не указывается, если образы контейнеров доступны анонимно.

■ **spec.policies.verifyImageSignatures.publicKeys**

массив строк

Обязательный параметр

Список публичных ключей Cosign.

■ **spec.policies.verifyImageSignatures.reference**

строка

Обязательный параметр

Адрес или шаблон адресов образов контейнеров.

Если содержит символ `*`, то считается шаблоном адресов образов контейнеров. Символ `*` может быть только один и только в конце. Например, для значения `company.registry.com/*` с указанными ключами и параметрами доступа будут проверяться любые образы контейнеров из хранилища `company.registry.com`.

Если не содержит символа `*`, то считается абсолютным адресом образа контейнера. Например, для значения `company.registry.com/nginx` с указанными ключами и параметрами доступа будет проверяться только образ `nginx` из хранилища `company.registry.com`.

Шаблон: `^[a-z0-9\.\-:@\/]*\*?$`

Примеры:

```
reference: docker.io/myuser/*
```

```
reference: ""
```

Custom Resources (от Gatekeeper)



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Mutation Custom Resources



Для мутационных хуков используется настройка `reinvocationPolicy: IfNeeded` в `MutatingWebhookConfiguration`. Подробнее в документации Kubernetes.

Reference

Представляют собой набор настраиваемых политик модификации ресурсов Kubernetes в момент их создания.

AssignMetadata

Reference

Позволяет изменять секцию `Metadata` ресурса.

На данный момент сервисом Gatekeeper разрешено только **добавление** объектов `labels` и `annotations`. Изменение существующих объектов не предусмотрено.

i В `spec.match.kinds` запрещено использовать `*`. При указании `*` мутация не применяется. Вместо этого необходимо явно перечислять целевые ресурсы (`kinds`) и их `apiGroups`.

Пример 1. Добавление лейбла `owner` со значением `admin` во всех неймспейсах:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: demo-annotation-owner
spec:
  match:
    scope: Namespaced
    location: "metadata.labels.owner"
    parameters:
      assign:
        value: "admin"
```

Пример 2. Добавление лейбла в конкретном пространстве имён и только на выбранные ресурсы:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: set-labels-<your_namespace>
spec:
  match:
    scope: Namespaced
    namespaces: [<"<your_namespace>">]
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"] # Запрещено использовать "*"
    location: "metadata.labels.<your_label_name>"
    parameters:
      assign:
        value: <your_label_value>
```

Assign

Позволяет изменять поля, за пределом секции `Metadata`.

Пример установки `imagePullPolicy` для всех контейнеров на `Always` во всех неймспейсах, кроме `system`:

```

apiVersion: mutations.gatekeeper.sh/v1
kind: Assign
metadata:
  name: demo-image-pull-policy
spec:
  applyTo:
    - groups: [""]
      kinds: ["Pod"]
      versions: ["v1"]
  match:
    scope: Namespaced
    kinds:
      - apiGroups: ["*"]
        kinds: ["Pod"]
    excludedNamespaces: ["system"]
  location: "spec.containers[name:*.imagePullPolicy]"
  parameters:
    assign:
      value: Always

```

ModifySet

Reference

Позволяет добавлять и удалять элементы из списка, например из списка аргументов для запуска контейнера.

Новые значения добавляются в конец списка.

Пример удаления аргумента `--alsologtostderr` из всех контейнеров в поде:

```

apiVersion: mutations.gatekeeper.sh/v1
kind: ModifySet
metadata:
  name: remove-err-logging
spec:
  applyTo:
    - groups: [""]
      kinds: ["Pod"]
      versions: ["v1"]
  location: "spec.containers[name: *].args"
  parameters:
    operation: prune
    values:
      fromList:
        - --alsologtostderr

```

AssignImage

Reference

Позволяет вносить изменения в параметр `image` ресурса.

Пример изменения параметра `image` на значение `my.registry.io/repo/app@sha256:abcde67890123456789abc345678901a`:

```

apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: AssignImage
metadata:
  name: assign-container-image
spec:
  applyTo:
    - groups: [ "" ]
      kinds: [ "Pod" ]
      versions: [ "v1" ]
  location: "spec.containers[name:].image"
  parameters:
    assignDomain: "my.registry.io"
    assignPath: "repo/app"
    assignTag: "@sha256:abcde67890123456789abc345678901a"
  match:
    source: "All"
    scope: Namespaced
    kinds:
      - apiGroups: [ "*" ]
        kinds: [ "Pod" ]

```

FAQ

Как настроить альтернативные решения по управлению политиками безопасности?

Для корректной работы DKP необходимы расширенные привилегии на запуск и работу полезной нагрузки системных компонентов. Если вместо модуля admission-policy-engine используется альтернативное решение по управлению политиками безопасности (например, Kyverno), необходима настройка исключений для следующих неймспейсов:

- kube-system;
- все неймспейсы с префиксом d8-* (например, d8-system).

Как настроить селекторы политик?

В OperationPolicy и SecurityPolicy поле spec.match определяет, на какие именно объекты (поды) в кластере будет распространяться политика. Оно обязательно должно присутствовать в конфигурации. Фильтрация выполняется путём комбинирования двух основных критериев: селектора подов (labelSelector) и селектора неймспейсов (namespaceSelector).

Если указаны оба селектора, политика применяется только к тем подам, которые **одновременно**:

- удовлетворяют условиям выбора подов;
- находятся в неймспейсах, прошедших фильтрацию.

Если какой-либо из селекторов не указан, соответствующая проверка не производится (будут использоваться все поды или неймспейсы).

spec.match.labelSelector - выбор подов

С помощью labelSelector задаются критерии отбора подов по их лейблам. Поддерживаются два взаимоисключающих способа:

- **matchLabels** – простая проверка на точное совпадение лейблов (ключ-значение). Под должен иметь все указанные лейблы.
- **matchExpressions** – гибкие выражения с операторами. Каждое выражение задаётся объектом с полями:
 - key (строка, обязательно) – имя лейбла.
 - operator (строка, обязательно) – одно из значений: In, NotIn, Exists, DoesNotExist.
 - values (массив строк) – список значений для операторов In / NotIn; для Exists / DoesNotExist не указывается.

Все элементы списка matchExpressions объединяются логическим **И** – под должен удовлетворять каждому выражению.

Примеры:


```

spec:
  match:
    labelSelector:
      matchLabels:
        app: nginx
        role: frontend

```

```

spec:
  match:
    labelSelector:
      matchExpressions:
        - key: tier
          operator: In
          values:
            - production
            - staging
        - key: monitoring
          operator: Exists

```

spec.match.namespaceSelector - выбор неймспейсов

Позволяет ограничить неймспейсы, в которых действует политика. Можно использовать комбинацию трёх фильтров:

- **matchNames** – явный список разрешённых неймспейсов. Если задан, политика действует только в перечисленных неймспейсах.
- **excludeNames** – список исключаемых неймспейсов. Политика будет действовать во всех неймспейсах, **кроме** указанных.
- **labelSelector** – селектор по лейблам самого объекта Namespace. Синтаксис полностью аналогичен селектору подов (`matchLabels` / `matchExpressions`). Фильтруются именно метаданные неймспейса.

Все заданные условия внутри `namespaceSelector` также объединяются логическим **И**. Рекомендуется не смешивать `matchNames` и `excludeNames` без явной необходимости – если указаны оба, результирующий набор вычисляется как `(matchNames ∩ все) \ excludeNames` (сначала берутся разрешённые, потом из них вычитаются исключённые).

Примеры:

```

spec:
  match:
    namespaceSelector:
      matchNames:
        - production
        - staging

```

```

spec:
  match:
    namespaceSelector:
      excludeNames:
        - kube-system
        - gatekeeper-system

```

```

spec:
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          team: backend
          environment: production

```

```

спес:
  match:
    namespaceSelector:
    labelSelector:
      matchExpressions:
        - key: compliance
          operator: In
          values:
            - pci
            - sox

```

Совместное использование селекторов

Наиболее типичный сценарий – ограничение и по подам, и по неймспейсам одновременно:

```

спес:
  match:
    labelSelector:
      matchLabels:
        app: payments
        version: v2
    namespaceSelector:
      labelSelector:
        matchLabels:
          env: prod
      excludeNames:
        - legacy-prod

```

Эта политика сработает для подов с лейблами `app=payments` и `version=v2`, которые находятся в неймспейсах, имеющих лейбл `env=prod`, за исключением неймспейса `legacy-prod`.

```

спес:
  match:
    labelSelector:
      matchExpressions:
        - key: security
          operator: NotIn
          values:
            - low
    namespaceSelector:
      matchNames:
        - frontend
        - backend

```

Здесь политика охватывает поды, у которых значение лейбла `security` не равно `low`, в неймспейсах `frontend` и `backend`.

Как расширить политики Pod Security Standards?

i Pod Security Standards реагируют на label `security.deckhouse.io/pod-policy: restricted` или `security.deckhouse.io/pod-policy: baseline`.

Чтобы расширить политику Pod Security Standards, добавив к существующим проверкам политики свои собственные, необходимо:

- создать шаблон проверки (`ConstraintTemplate`);
- привязать его к политике `restricted` или `baseline`.

Пример шаблона для проверки адреса репозитория образа контейнера:

```

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }

```

Пример привязки проверки к политике `restricted`:

```

apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"

```

Пример демонстрирует настройку проверки адреса репозитория в поле `image` у всех подов, создающихся в неймспейсах, имеющих label `security.deckhouse.io/pod-policy: restricted`. Если адрес в поле `image` создаваемого пода начинается не с `mycompany.registry.com`, под создан не будет.

Подробнее о шаблонах и языке политик можно узнать в документации Gatekeeper.

Больше примеров описания проверок для расширения политики можно найти в библиотеке Gatekeeper.

Как включить одну или несколько политик Pod Security Standards, не отключая весь набор?

Чтобы применить только нужные политики безопасности, не отключая весь предустановленный набор:

1. Добавьте в нужный неймспейс лейбл: `security.deckhouse.io/pod-policy: privileged`, чтобы отключить встроенный набор политик.
2. Создайте ресурс SecurityPolicy, соответствующий уровню `baseline` или `restricted`. В секции `policies` укажите только необходимые вам настройки.
3. Добавьте в неймспейс дополнительный лейбл, который будет соответствовать селектору `namespaceSelector` в SecurityPolicy. В примерах ниже это `security-policy.deckhouse.io/baseline-enabled: "true"` либо `security-policy.deckhouse.io/restricted-enabled: "true"`

i Поле `allowedHostPaths` определяет список разрешённых префиксов путей для монтирования `hostPath`. Если указан пустой список (`[]`), использование `hostPath` запрещено для всех путей. Если поле отсутствует, ограничения не применяются.

SecurityPolicy, соответствующая `baseline`:

```

apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - AUDIT_WRITE
      - CHOWN
      - DAC_OVERRIDE
      - FOWNER
      - FSETID
      - KILL
      - MKNOD
      - NET_BIND_SERVICE
      - SETFCAP
      - SETGID
      - SETPCAP
      - SETUID
      - SYS_CHROOT
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    selinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
        - undefined
        - ''
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/baseline-enabled: "true"

```

SecurityPolicy, соответствующая restricted:

```

apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: restricted
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
    requiredDropCapabilities:
      - ALL
    runAsUser:
      rule: MustRunAsNonRoot
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/restricted-enabled: "true"

```

Что, если несколько политик (операционных или безопасности) применяются на один объект?

В этом случае необходимо, чтобы конфигурация объекта соответствовала всем политикам, которые на него распространяются.

Например, рассмотрим две следующие политики безопасности:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: foo
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
      - MKNOD
---
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: bar
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    requiredDropCapabilities:
      - NET_BIND_SERVICE
```

Тогда для выполнения требований приведенных политик безопасности в спецификации контейнера нужно указать:

```
securityContext:
  capabilities:
    drop:
      - MKNOD
      - NET_BIND_SERVICE
  readOnlyRootFilesystem: true
```

Проверка подписи образов

В модуле реализована функция проверки подписи образов контейнеров, подписанных с помощью инструмента Cosign. Подробнее о подписании и проверке образов контейнеров можно узнать в документации DKP.

Как запретить удаление узла без лейбла

i Операции DELETE обрабатываются Gatekeeper по умолчанию.

Можно создать собственную политику Gatekeeper, запрещающую удаление узла без специального лейбла. Пример ниже использует `oldObject` для проверки лейблов удаляемого узла:

```

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Удаление Node запрещено. Добавьте лейбл %q=%q.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
      ---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"

```

Как запретить операции `kubectl exec` и `kubectl attach` в определённые поды?

Вебхук модуля `admission-policy-engine` направляет запросы `CONNECT` для `Pods/exec` и `Pods/attach` через Gatekeeper. Это позволяет создавать пользовательские политики для разрешения или запрета операций `kubectl exec` и `kubectl attach`.

Встроенная политика для подов с `heritage: deckhouse`

Для защиты системных компонентов под управлением Deckhouse в модуле `admission-policy-engine` предусмотрена встроенная политика `D8DenyExecHeritage`, которая запрещает выполнение операций `kubectl exec` и `kubectl attach` во все поды с лейблом `heritage: deckhouse`.

Политика не распространяется на следующих пользователей, которым разрешены операции `kubectl exec` и `kubectl attach` в поды с лейблом `heritage: deckhouse`:

- `system:sudouser`;
- сервисные аккаунты из неймспейсов `d8-*` (`system:serviceaccount:d8-*`);
- сервисные аккаунты из неймспейсов `kube-*` (`system:serviceaccount:kube-*`).

Пример пользовательской политики

Вы можете создать собственную политику Gatekeeper для запрета операций `kubectl exec` и `kubectl attach` в определённых неймспейсах. В примере ниже используются `input.review.operation` и `input.review.resource.resource` для проверки операций `CONNECT`:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource предпочтительнее, но на всякий случай падаем в subResource
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource", input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
        }

        violation[{"msg": msg}] {
          is_connect
          is_exec_or_attach
          is_forbidden_namespace
          msg := sprintf("Exec/attach запрещён в неймспейсе %q", [input.review.namespace])
        }
      ---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
  name: deny-exec-in-namespaces
spec:
```

```

enforcementAction: deny
match:
  kinds:
    - apiGroups: ["*"]
      kinds: ["*"]
  scope: Namespaced
parameters:
  forbiddenNamespaces:
    - production
    - staging

```

Ключевые данные и проверки, доступные при валидации операций `CONNECT`:

- Используйте `input.review.operation == "CONNECT"` для проверки операций `CONNECT`.
- Информация о пользователе доступна в `input.review.userInfo.username` и `input.review.userInfo.groups`.
- Неймспейс доступен в `input.review.namespace`.

Модуль cert-manager

Устанавливает надежную и высокодоступную инсталляцию cert-manager версии 1.20.0.

При установке модуля автоматически учитываются особенности кластера:

- компонент (webhook), к которому обращается `kube-apiserver`, устанавливается на master-узлы;
- в случае недоступности вебхука производится временное удаление `apiservice`, чтобы недоступность `cert-manager` не блокировала работу кластера.

Обновление самого модуля происходит в автоматическом режиме, в том числе с миграцией ресурсов cert-manager.

Возможности модуля cert-manager (с учетом внесенных изменений)

Модуль обеспечивает использование всех возможностей оригинального cert-manager, в том числе:

- заказ сертификатов во всех поддерживаемых источниках, таких как *Let's Encrypt*, *HashiCorp Vault*, *Venafi*;
- выпуск самоподписанных сертификатов;
- поддержку актуальности сертификатов, автоматический перевыпуск и т. д.

Изменения в оригинальный cert-manager были внесены, чтобы поды `cm-acme-http-solver` могли выполняться на master-узлах и выделенных узлах.

Мониторинг

Модуль обеспечивает экспорт метрик в Prometheus для мониторинга:

- срока действия сертификатов;
- корректности перевыпуска сертификатов.

Роли доступа к ресурсам

В модуле предопределены несколько продуманных ролей для удобного доступа к ресурсам:

- `User` – доступ на чтение к ресурсам Certificate и Issuer в доступных ему namespace, а также к глобальным ClusterIssue;
- `Editor` – управление ресурсами Certificate и Issuer в доступных ему namespace;
- `ClusterEditor` – управление ресурсами Certificate и Issuer в любых namespace;
- `SuperAdmin` – управление внутренними служебными объектами.

Настройки

У модуля нет обязательных настроек.

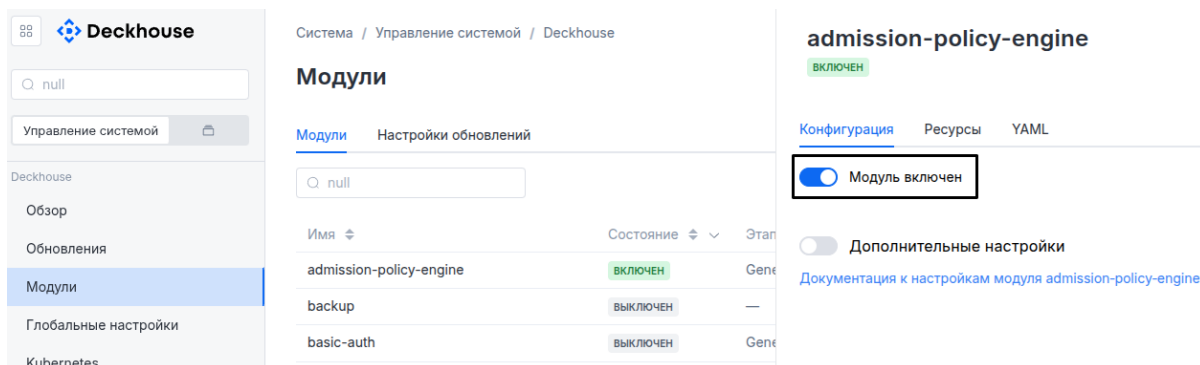
В модуле реализовано 6 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cert-manager`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cert-manager`:

```
d8 system module enable cert-manager
```

- **С помощью ModuleConfig `cert-manager`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cert-manager` (создайте его, при необходимости).

Пример манифеста для включения модуля `cert-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  enabled: true
```

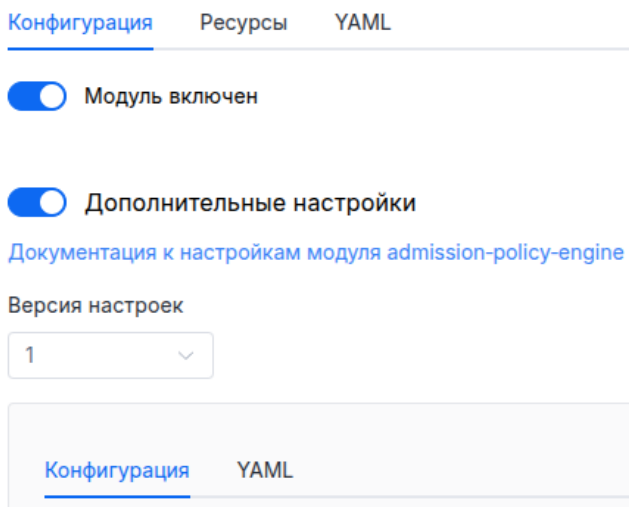
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cert-manager`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `cert-manager` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cert-manager`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cert-manager` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cert-manager
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cert-manager`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cert-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.cleanupOrphanSecrets**

булевый

Удалять Secret с сертификатом автоматически, если соответствующий ресурс Certificate удален из кластера.

По умолчанию: `false`

Примеры:

```
cleanupOrphanSecrets: true
```

```
cleanupOrphanSecrets: false
```

- **settings.cloudDNSServiceAccount**

строка

Service Account для Google Cloud из того же проекта с ролью Администратора DNS.

Пример:

```
cloudDNSServiceAccount:
  eyJzYSI6ICJhYmNkZWZnaEBzZXJ2aWNlYWVudC5jb20iLCAicHJvamVjdF9pZCI6ImFhYWVhIn0=
```

- **settings.cloudflareAPIToken**

строка

API Tokens позволяют использовать ключи, привязанные к определенным DNS-зонам.

Рекомендуется использовать API Tokens для более высокой безопасности, поскольку они имеют более строгие разрешения и могут быть легко отозваны.

Способ проверки того, что домены, указанные в ресурсе Certificate, для которых заказывается сертификат, находятся под управлением `cert-manager` у DNS-провайдера Cloudflare. Проверка происходит путем добавления специальных TXT-записей для домена ACME DNS01 Challenge Provider.

Пример:

```
cloudflareAPIToken: token
```

- **settings.cloudflareEmail**
строка

Почтовый ящик проекта, на который выдавались доступы для управления Cloudflare.

Пример:

```
cloudflareEmail: example@example.com
```

- **settings.cloudflareGlobalAPIKey**
строка

Cloudflare Global API key для управления DNS-записями.

Способ проверки того, что домены, указанные в ресурсе Certificate, для которых заказывается сертификат, находятся под управлением cert-manager у DNS-провайдера Cloudflare. Проверка происходит путем добавления специальных TXT-записей для домена ACME DNS01 Challenge Provider.

Пример:

```
cloudflareGlobalAPIKey: key
```

- **settings.digitalOceanCredentials**
строка

Access Token от Digital Ocean API, который можно создать в разделе `API`.

Пример:

```
digitalOceanCredentials: creds
```

- **settings.disableLetsencrypt**
булевый

Не создавать ClusterIssuer `letsencrypt` и `letsencrypt-staging` в кластере (если `true`).

Примеры:

```
disableLetsencrypt: true
```

```
disableLetsencrypt: false
```

- **settings.email**
строка

Email проекта, на который LetsEncrypt будет слать уведомления.

Пример:

```
email: example@example.com
```

- **settings.enableCAInjector**
булевый

Включить CAInjector. Он необходим только для инъекции CA-сертификатов в `ValidatingWebhookConfiguration`, `MutatingWebhookConfiguration`, `CustomResourceDefinition` и `APIService`. Deckhouse не использует CAInjector, поэтому включать нужно только в том случае, если вы используете в своих сервисах пользовательские инъекции CA.

По умолчанию: `false`

Примеры:

```
enableCAInjector: true
```

```
enableCAInjector: false
```

- **settings.enableListenerSet**
булевый

Включить поддержку ListenerSet в cert-manager через feature gate `ListenerSets`.

Также включает интеграцию ListenerSet для Gateway API, если включена интеграция Gateway API.

По умолчанию: `false`

Примеры:

```
enableListenerSet: true
```

```
enableListenerSet: false
```

- **settings.ingressClassHttp01**
строка

Имя `ingressClass` используемого для подтверждения владения доменом методом ACME HTTP-01. Если параметр не указан, то используется `ingressClass` по умолчанию.

Пример:

```
ingressClassHttp01: nginx
```

- **settings.letsencrypt**
объект

Настройки для ClusterIssuer `letsencrypt`.

- **settings.letsencrypt.resources**
объект

Настройки ресурсов (requests/limits по CPU и памяти) для подов, создаваемых ClusterIssuer `letsencrypt` для процесса подтверждения владения доменом методом ACME HTTP-01. Структура, аналогичная `spec.containers[].resources` пода Kubernetes.

Пример:

```
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 250m
    memory: 256Mi
```

- **settings.letsencrypt.resources.limits**
объект

- **settings.letsencrypt.resources.limits.cpu**
строка

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.cpu` в Kubernetes.

По умолчанию: `50m`

Шаблон: `^[0-9]+m?$`

- **settings.letsencrypt.resources.limits.memory**
строка or число

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.memory` в Kubernetes.

По умолчанию: `64Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencrypt.resources.requests**
объект

Resource requests and limits (CPU and memory) for pods that create the `letsencrypt` ClusterIssuer for http01 challenges

- **settings.letsencrypt.resources.requests.cpu**
строка

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.cpu` в Kubernetes.

По умолчанию: `10m`

Шаблон: `^[0-9]+m?$`

- **settings.letsencrypt.resources.requests.memory**

строка or число

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.memory` в Kubernetes.

По умолчанию: `32Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencryptStaging**

объект

Настройки для ClusterIssuer `letsencrypt-staging`.

- **settings.letsencryptStaging.resources**

объект

Настройки ресурсов (requests/limits по CPU и памяти) для подов, создаваемых ClusterIssuer `letsencrypt-staging` для процесса подтверждения владения доменом методом ACME HTTP-01. Структура, аналогичная `spec.containers[].resources` пода Kubernetes.

Пример:

```
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 250m
    memory: 256Mi
```

- **settings.letsencryptStaging.resources.limits**

объект

- **settings.letsencryptStaging.resources.limits.cpu**

строка

Обязательный параметр

Аналогично параметру `spec.containers[].resources.limits.cpu` в Kubernetes.

По умолчанию: `50m`

Шаблон: `^[0-9]+m?$`

- **settings.letsencryptStaging.resources.limits.memory**

строка or число

Обязательный параметр

Аналогично параметру `spec.containers[].resources.limits.memory` в Kubernetes.

По умолчанию: `64Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencryptStaging.resources.requests**

объект

- **settings.letsencryptStaging.resources.requests.cpu**

строка

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.cpu` в Kubernetes.

По умолчанию: `10m`

Шаблон: `^[0-9]+m?$`

- **settings.letsencryptStaging.resources.requests.memory**

строка or число

Обязательный параметр

Аналогично параметру `spec.containers[].resources.requests.memory` в Kubernetes.

По умолчанию: 32Mi

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.maxConcurrentChallenges**

целочисленный

Максимальное количество одновременных Challenges в статусе `Processing`

Допустимые значения: `0 <= X`

Пример:

```
maxConcurrentChallenges: 25
```

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
nodeSelector:
  has-gpu: 'true'
```

- **settings.recursiveSettings**

объект

Параметры использования рекурсивных DNS-серверов, которые применяются для проверки существования записи в DNS, перед запуском процесса подтверждения владения доменом методом ACME DNS-01.

Пример:

```
recursiveSettings:
  nameservers:
    - 8.8.8.8:53
    - https://1.1.1.1
  useOnlyRecursive: true
```

- **settings.recursiveSettings.nameservers**

массив строк

Список рекурсивных DNS-серверов (IP:порт или HTTPS-URL).

- **Элемент массива**

строка

Шаблон: `^(?:[0-9]{1,3}(?:\.[0-9]{1,3}){3}:\d{1,5}|https?:\/\/[^\s]+)$`

- **settings.recursiveSettings.useOnlyRecursive**

булевый

Если значение `true`, для проверки существования записи в DNS перед запуском процесса подтверждения владения доменом будут использоваться только указанные рекурсивные DNS-серверы.

По умолчанию: `false`

- **settings.route53AccessKeyID**

строка

Access Key ID пользователя с необходимыми правами.

Amazon Route53 IAM Policy для управления доменными записями домена.

Пример:

```
route53AccessKeyID: key_id
```

- **settings.route53SecretAccessKey**

строка

Secret Access Key пользователя с необходимыми правами для управления доменными записями домена.

Пример:


```
route53SecretAccessKey: secret
```

◦ **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
tolerations:
- key: dedicated.deckhouse.io
  operator: Equal
  value: cert-manager
```

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Custom resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Для запроса сертификатов модуль использует стандартный custom resource cert-manager — Certificate.

Примеры конфигурации

Пример заказа сертификата

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com # Имя сертификата, через него потом можно смотреть
  namespace: default статус.
spec:
  secretName: example-com-tls # Название Secret'a, в который положить приватный
  issuerRef: # Ссылка на "выдаватель" сертификатов, см. подробнее
    kind: ClusterIssuer
    name: letsencrypt
  commonName: example.com # Основной домен сертификата.
  dnsNames: # Дополнительные домены сертификата (как минимум одно
    - www.example.com DNS-имя или IP-адрес должны быть указаны).
    - admin.example.com
```

При этом:

- создается отдельный Ingress-ресурс на время прохождения challenge'a (соответственно, аутентификация и whitelist основного Ingress не будут мешать);
- можно заказать один сертификат на несколько Ingress-ресурсов (и он не отвалится при удалении того, в котором была аннотация `tls-acme`);
- можно заказать сертификат с дополнительными именами (как в примере);
- можно валидировать разные домены, входящие в один сертификат, через разные Ingress-контроллеры.

Подробнее можно прочитать в документации cert-manager.

Заказ wildcard-сертификата с DNS в Cloudflare

1. Получим `Global API Key` и `Email Address`:

- Заходим на страницу: <https://dash.cloudflare.com/profile>.
- В самом верху страницы написана ваша почта под `Email Address`.
- В самом низу страницы жмем на кнопку `View` напротив `Global API Key`.

В результате этого мы получаем ключ для взаимодействия с API Cloudflare и почту, на которую зарегистрирован аккаунт.

2. Редактируем настройки модуля `cert-manager` и добавляем такую секцию:

```
settings:
  cloudflareGlobalAPIKey: APIkey
  cloudflareEmail: some@mail.somedomain
```

или

```
settings:
  cloudflareAPIToken: some-token
  cloudflareEmail: some@mail.somedomain
```

После этого Deckhouse автоматически создаст `ClusterIssuer` и `Secret` для Cloudflare в namespace `d8-cert-manager`.

- Конфигурация с помощью `APIToken` является рекомендуемой и более безопасной.

3. Создаем `Certificate` с проверкой с помощью провайдера Cloudflare. Данная возможность появится только при указании настройки `cloudflareGlobalAPIKey` и `cloudflareEmail` в Deckhouse:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: cloudflare
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
    - "*.domain.com"
```

4. Создаем `Ingress`:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  ingressClassName: nginx
  rules:
    - host: "*.domain.com"
      http:
        paths:
          - backend:
              service:
                name: svc-web
                port:
                  number: 80
            path: /
  tls:
    - hosts:
        - "*.domain.com"
      secretName: tls-wildcard
```

Заказ wildcard-сертификата с DNS в Route53

1. Создаем пользователя с необходимыми правами.

- Заходим на страницу управления политиками. Создаем политику с такими правами:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "route53:GetChange",
      "Resource": "arn:aws:route53::change/*"
    },
    {
      "Effect": "Allow",
      "Action": "route53:ChangeResourceRecordSets",
      "Resource": "arn:aws:route53::hostedzone/*"
    },
    {
      "Effect": "Allow",
      "Action": "route53:ListHostedZonesByName",
      "Resource": "*"
    }
  ]
}
```

- Заходим на страницу управления пользователями. Создаем пользователя с созданной ранее политикой.

2. Редактируем настройки модуля cert-manager и добавляем такую секцию:

```
settings:
  route53AccessKeyID: <ACCESS_KEY_ID>
  route53SecretAccessKey: <SECRET_ACCESS_KEY>
```

После этого Deckhouse автоматически создаст ClusterIssuer и Secret для route53 в namespace `d8-cert-manager`.

3. Создаем Certificate с проверкой с помощью провайдера route53. Данная возможность появится только при указании настроек `route53AccessKeyID` и `route53SecretAccessKey` в Deckhouse:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: route53
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
  - "*.domain.com"
```

Заказ wildcard-сертификата с DNS в Google

1. Создаем ServiceAccount с необходимой ролью:

- Заходим на страницу управления политиками.
- Выбираем нужный проект.
- Создаем ServiceAccount с желаемым названием, например `dns01-solver`.
- Заходим в созданный ServiceAccount.
- Создаем ключ по кнопке «Добавить ключ».
- Будет скачан `.json`-файл с данными ключа имени.
- Закодируем полученный файл в строку **base64**:

```
base64 project-209317-556c656b81c4.json
```

2. Сохраняем полученную **base64**-строку в параметр модуля `cloudDNSServiceAccount`.

После этого Deckhouse автоматически создаст ClusterIssuer и Secret для cloudDNS в namespace `d8-`

`cert-manager`.

3. Создаем Certificate с валидацией через cloudDNS:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: clouddns
    kind: ClusterIssuer
  dnsNames:
    - "*.domain.com"
```

Заказ self-signed-сертификата

Все еще проще, чем с LetsEncrypt. Просто меняем `letsencrypt` на `selfsigned`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com # Имя сертификата, через него потом можно смотреть статус.
  namespace: default
spec:
  secretName: example-com-tls # Название Secret'a, в который положить приватный ключ и сертификат.
  issuerRef:
    kind: ClusterIssuer # Ссылка на ClusterIssuer.
    name: selfsigned
  commonName: example.com # Основной домен сертификата.
  dnsNames: # Дополнительные домены сертификата. Требуется, как минимум, дублирование записи из commonName.
    - example.com
    - www.example.com
    - admin.example.com
```

i Пример создания самоподписанного сертификата вручную, без использования утилиты `cert-manager`, доступен в разделе FAQ.

FAQ

Какие виды сертификатов поддерживаются?

На данный момент модуль устанавливает следующие `ClusterIssuer`:

- `letsencrypt`
- `letsencrypt-staging`
- `selfsigned`
- `selfsigned-no-trust`

Если требуется поддержка других типов сертификатов, вы можете добавить их самостоятельно.

Как добавить дополнительный `ClusterIssuer`?

В каких случаях требуется дополнительный `ClusterIssuer`?

В стандартной поставке присутствуют `ClusterIssuer`, издающие либо сертификаты из доверенного публичного удостоверяющего центра Let's Encrypt, либо самоподписанные сертификаты.

Чтобы издать сертификаты на доменное имя через Let's Encrypt, сервис требует осуществить подтверждение владения доменом. `Cert-manager` поддерживает несколько методов для такого подтверждения при использовании `ACME` (Automated Certificate Management Environment):

- `HTTP-01` — `cert-manager` создаст временный Pod в кластере, который будет слушать на определенном URL для подтверждения владения доменом. Для его работы необходимо иметь

возможность направлять внешний трафик на этот Pod, обычно через `Ingress`.

- `DNS-01` — `cert-manager` делает TXT-запись в DNS для подтверждения владения доменом. У `cert-manager` есть встроенная поддержка популярных провайдеров DNS: AWS Route53, Google Cloud DNS, Cloudflare и т.д. Полный перечень доступен в документации `cert-manager`.



Метод `HTTP-01` не поддерживает выпуск wildcard-сертификатов.

Поставляемые `ClusterIssuers`, издающие сертификаты через Let's Encrypt, делятся на два типа:

1. `ClusterIssuer`, специфичные для используемого облачного провайдера. Добавляются автоматически, при заполнении настроек модуля связанных с облачным провайдером. Поддерживают метод `DNS-01`.
 - `clouddns`
 - `cloudflare`
 - `digitalocean`
 - `route53`
2. `ClusterIssuer` использующие метод `HTTP-01`. Добавляются автоматически, если их создание не отключено в настройках модуля.
 - `letsencrypt`
 - `letsencrypt-staging`

Таким образом, дополнительный `ClusterIssuer` может потребоваться в случаях издания сертификатов:

1. В удостоверяющем центре (УЦ), отличном от Let's Encrypt (в т.ч. в приватном). Поддерживаемые УЦ доступны в документации `cert-manager`
2. Через Let's Encrypt с помощью метода `DNS-01` через сторонний провайдер.

Как добавить дополнительный `ClusterIssuer`, использующий Let's Encrypt и метод подтверждения `DNS-01`?

Для подтверждения владения доменом через Let's Encrypt с помощью метода `DNS-01` требуется настроить возможность создания TXT-записей в публичном DNS.

У `cert-manager` есть поддержка механизмов для создания TXT-записей в некоторых популярных DNS: AzureDNS, Cloudflare, Google Cloud DNS и т.д.

Полный перечень доступен в документации `cert-manager`.

Модуль автоматически создает `ClusterIssuer` поддерживаемых облачных провайдеров, при заполнении настроек модуля связанных с используемым облаком.

При необходимости можно создать такие `ClusterIssuer` самостоятельно.

Пример использования AWS Route53 доступен в разделе Как защитить учетные данные `cert-manager`. Актуальный перечень всех возможных для создания `ClusterIssuer` доступен в шаблонах модуля.

Использование сторонних DNS-провайдеров реализуется через метод `webhook`.

Когда `cert-manager` выполняет вызов ACME `DNS-01`, он отправляет запрос на вебхук-сервер, который затем выполняет нужные операции для обновления записи DNS.

При использовании данного метода требуется разместить сервис, который будет обрабатывать хук и осуществлять создание TXT-записи в DNS-провайдере.

В качестве примера рассмотрим использование сервиса `Yandex Cloud DNS`.

1. Для обработки вебхука предварительно разместите в кластере сервис `Yandex Cloud DNS ACME webhook` согласно официальной документации.
2. Затем создайте ресурс `ClusterIssuer`:

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: yc-clusterissuer
  namespace: default
spec:
  acme:
    # Вы должны заменить этот адрес электронной почты на свой собственный.
    # Let's Encrypt будет использовать его, чтобы связаться с вами по поводу истекающих
    # сертификатов и вопросов, связанных с вашей учетной записью.
    email: your@email.com
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      # Ресурс секретов, который будет использоваться для хранения закрытого ключа
      # аккаунта.
      name: secret-ref
    solvers:
      - dns01:
          webhook:
            config:
              # Идентификатор папки, в которой расположена DNS-зона
              folder: <your folder ID>
              # Это секрет, используемый для доступа к учетной записи сервиса
              serviceAccountSecretRef:
                name: cert-manager-secret
                key: iamkey.json
              groupName: acme.cloud.yandex.com
              solverName: yandex-cloud-dns

```

Как добавить дополнительный **Issuer** и **ClusterIssuer**, использующий HashiCorp Vault для выпуска сертификатов?

Для выпуска сертификатов с помощью HashiCorp Vault, можете использовать инструкцию.

После конфигурации PKI и включения авторизации в Kubernetes выполните следующие шаги:

- Создайте **ServiceAccount**:

```
d8 k create serviceaccount issuer
```

- Создайте **Secret** с токеном для этой учётной записи. Начиная с Kubernetes 1.24, секрет для **ServiceAccount** не создаётся автоматически:

```

d8 k apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: issuer
  annotations:
    kubernetes.io/service-account.name: issuer
type: kubernetes.io/service-account-token
EOF

```

- Создайте **Issuer**:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    # Если Vault разворачивался по вышеуказанной инструкции, в этом месте в инструкции
    # опечатка.
    server: http://vault.default.svc.cluster.local:8200
    # Указывается на этапе конфигурации PKI.
    path: pki/sign/example-dot-com
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: issuer
          key: token
EOF
```

- Создать ресурс `Certificate` для получения TLS-сертификата, подписанного CA Vault:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # Домены указываются на этапе конфигурации PKI в Vault.
  commonName: www.example.com
  dnsNames:
    - www.example.com
EOF
```

Как добавить `ClusterIssuer`, использующий свой или промежуточный CA для заказа сертификатов?

Для использования собственного или промежуточного CA:

- Сгенерируйте сертификат (при необходимости):

```
openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
```

- В пространстве имён `d8-cert-manager` создайте секрет, содержащий данные файлов сертификатов. Пример создания секрета с помощью команды `d8 k`:

```
d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --
cert="rootCACert.pem"
```

Пример создания секрета из YAML-файла (содержимое файлов сертификатов должно быть закодировано в Base64):

```

apiVersion: v1
data:
  tls.crt: <результат команды `cat rootCACert.pem | base64 -w0`>
  tls.key: <результат команды `cat rootCAKey.pem | base64 -w0`>
kind: Secret
metadata:
  name: internal-ca-key-pair
  namespace: d8-cert-manager
type: Opaque

```

Имя секрета может быть любым.

- Создайте `ClusterIssuer` из созданного секрета:

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: inter-ca
spec:
  ca:
    secretName: internal-ca-key-pair # Имя созданного секрета.

```

Имя `ClusterIssuer` также может быть любым.

Теперь можно использовать созданный `ClusterIssuer` для получения сертификатов для всех компонентов Deckhouse или конкретного компонента.

Например, чтобы использовать `ClusterIssuer` для получения сертификатов для всех компонентов Deckhouse, укажите его имя в глобальном параметре `clusterIssuerName` (`d8 k edit mc global`):

```

spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
          mode: CertManager
          publicDomainTemplate: '%s.<public_domain_template>'
version: 1

```

Как защитить учетные данные `cert-manager`?

Если вы не хотите хранить учетные данные конфигурации Deckhouse (например, по соображениям безопасности), можете создать свой собственный `ClusterIssuer` / `Issuer`.

Пример создания собственного `ClusterIssuer` для сервиса `route53`:

- Создайте `Secret` с учетными данными:

```

d8 k apply -f - <<EOF
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: route53
  namespace: default
data:
  secret-access-key: MY-AWS-ACCESS-KEY-TOKEN
EOF

```

- Создайте простой `ClusterIssuer` со ссылкой на этот `Secret`:


```

d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: route53
  namespace: default
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: route53-tls-key
    solvers:
      - dns01:
          route53:
            region: us-east-1
            accessKeyID: MY-AWS-ACCESS-KEY-ID
            secretAccessKeySecretRef:
              name: route53
              key: secret-access-key
EOF

```

- Закажите сертификаты как обычно, используя созданный `ClusterIssuer`:

```

d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: route53
  commonName: www.example.com
  dnsNames:
    - www.example.com
EOF

```

Работает ли аннотация TLS-асме?

Да, работает. Специальный компонент `cert-manager-ingress-shim` видит эти аннотации и на их основании автоматически создает ресурсы `Certificate` (в тех же namespaces, что и Ingress-ресурсы с аннотациями).



При использовании аннотации ресурс `Certificate` создаётся «прилинкованным» к существующему Ingress-ресурсу, и для прохождения Challenge не создаётся отдельный Ingress, а вносятся дополнительные записи в существующий. Это означает, что если на основном Ingress'е настроена аутентификация или whitelist — ничего не выйдет. Лучше не использовать аннотацию и переходить на ресурс `Certificate`.



При переходе с аннотации на ресурс `Certificate` удалите ресурс `Certificate`, который был создан по аннотации. Иначе по обоим ресурсам `Certificate` будет обновляться один Secret, и это может привести к достижению лимита запросов Let's Encrypt.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/tls-acme: "true"          # Аннотация.
  name: example-com
  namespace: default
spec:
  ingressClassName: nginx
  rules:
    - host: example.com
      http:
        paths:
          - backend:
              service:
                name: site
                port:
                  number: 80
              path: /
              pathType: ImplementationSpecific
    - host: www.example.com                  # Дополнительный домен.
      http:
        paths:
          - backend:
              service:
                name: site
                port:
                  number: 80
              path: /
              pathType: ImplementationSpecific
    - host: admin.example.com                # Еще один дополнительный домен.
      http:
        paths:
          - backend:
              service:
                name: site
                port:
                  number: 80
              path: /
              pathType: ImplementationSpecific
  tls:
    - hosts:
        - example.com
        - www.example.com                  # Дополнительный домен.
        - admin.example.com                # Еще один дополнительный домен.
      secretName: example-com-tls          # Имя для Certificate и Secret.

```

Как посмотреть состояние сертификата?

```
d8 k -n default describe certificate example-com
...
Status:
  Acme:
    Authorizations:
      Account: https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:  example.com
      Uri:     https://acme-
v01.api.letsencrypt.org/acme/challenge/qJA9MGCZnUnVjAgxhox0NvDnKAsPatRILJ4n0LJ7MMY/4062050823
      Account: https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:  admin.example.com
      Uri:     https://acme-v01.api.letsencrypt.org/acme/challenge/pW2tFKLBDTl12Gx8UBqmEl846x5W-
YpBs8a4HqstJK8/4062050808
      Account: https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:  www.example.com
      Uri:     https://acme-v01.api.letsencrypt.org/acme/challenge/LaZJMM9_0KcTYbETHjt3oLtwgpkNfbHVDl8Dz-
yypx8/4062050792
    Conditions:
      Last Transition Time: 2018-04-02T18:01:04Z
      Message:             Certificate issued successfully
      Reason:              CertIssueSuccess
      Status:              True
      Type:                Ready
  Events:
    Type      Reason              Age           From              Message
    ----      -
    Normal    PrepareCertificate    1m           cert-manager-controller    Preparing certificate with issuer
    Normal    PresentChallenge     1m           cert-manager-controller    Presenting http-01 challenge for
domain example.com
    Normal    PresentChallenge     1m           cert-manager-controller    Presenting http-01 challenge for
domain www.example.com
    Normal    PresentChallenge     1m           cert-manager-controller    Presenting http-01 challenge for
domain admin.example.com
    Normal    SelfCheck            1m           cert-manager-controller    Performing self-check for domain
admin.example.com
    Normal    SelfCheck            1m           cert-manager-controller    Performing self-check for domain
example.com
    Normal    SelfCheck            1m           cert-manager-controller    Performing self-check for domain
www.example.com
    Normal    ObtainAuthorization   55s          cert-manager-controller    Obtained authorization for domain
example.com
    Normal    ObtainAuthorization   54s          cert-manager-controller    Obtained authorization for domain
admin.example.com
    Normal    ObtainAuthorization   53s          cert-manager-controller    Obtained authorization for domain
www.example.com
```

Как получить список сертификатов?

```
d8 k get certificate --all-namespaces
```

NAMESPACE	NAME	AGE
default	example-com	13m

Что делать, если появляется ошибка: CAA record does not match issuer?

Если `cert-manager` не может заказать сертификаты с ошибкой:

```
CAA record does not match issuer
```

то необходимо проверить CAA (Certificate Authority Authorization) DNS-запись у домена, для которого заказывается сертификат. Если вы хотите использовать Let's Encrypt-сертификаты, у домена должна быть CAA-запись: `issue "letsencrypt.org"`. Подробнее про CAA можно почитать в документации Let's Encrypt.

Модуль chrony

Обеспечивает синхронизацию времени на всех узлах кластера с помощью утилиты `chrony`.

Как работает

Модуль запускает `chrony` агенты на всех узлах кластера. По умолчанию используется NTP сервер `pool.ntp.org`. NTP сервер можно изменить через настройки модуля. Для просмотра используемых NTP серверов можно воспользоваться командой:

```
d8 k exec -it -n d8-chrony ds/chrony-master -- chronyc -N sources
```

Пример вывода:

MS	Name/IP address	Stratum	Poll	Reach	LastRx	Last sample
^*	pool.ntp.org.	2	10	377	171	-502us[-909us] +/- 5388us
^-	pool.ntp.org.	2	10	377	666	-5317us[-5698us] +/- 103ms
^+	pool.ntp.org.	2	10	377	938	-201us[-567us] +/- 5346us
^+	pool.ntp.org.	2	10	377	843	-159us[-530us] +/- 12ms

i ^+ - комбинируемый NTP сервер(chrony комбинирует информацию из combined серверов для уменьшения неточностей);
 ^* - текущий NTP сервер;
 ^- - некомбинируемый NTP-сервер.
 Подробную информацию о значениях столбцов можно получить с помощью флага -v: `chronyc sources -v`.

chrony агенты на мастер узлах и на остальных узлах имеют одно главное отличие - на всех узлах, которые не являются мастерами, в списке NTP серверов находятся не только NTP сервера из `module config`, но и адреса всех мастер узлов кластера.

Таким образом, агенты на мастер узлах синхронизируют время только из списка хостов, указанных в `module config` (по умолчанию с `pool.ntp.org`). А агенты на остальных узлах синхронизируют время со списком NTP серверов из `module config` плюс с `chrony` агентов на мастер узлах.

Это сделано для того, чтобы в случае недоступности NTP серверов, указанных в `module config`, время синхронизировалось с мастер узлами.

Настройки

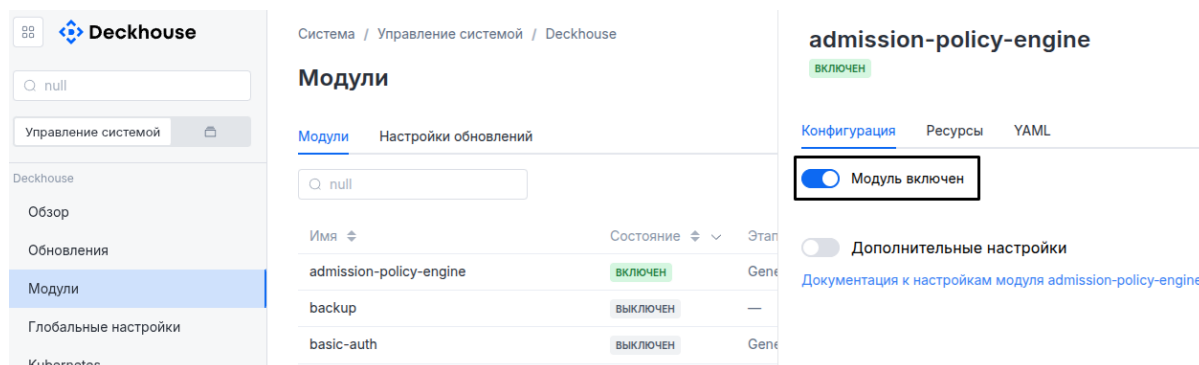
В модуле реализовано 2 алерта.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `chrony`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `chrony`:

```
d8 system module enable chrony
```

- **С помощью ModuleConfig `chrony`.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig chrony` (создайте его, при необходимости).

Пример манифеста для включения модуля `chrony`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true

```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `chrony`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект `ModuleConfig` `chrony` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `chrony`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig` `chrony` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc chrony
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig` `chrony`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig` `chrony`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.ntpServers**

массив строк

Список NTP-серверов для синхронизации.

По умолчанию: `["pool.ntp.org"]`

Примеры:

```
ntpServers:
- pool.ntp.org
```

```
ntpServers:
- pool.ntp.org
- ntp.ubuntu.com
- time.google.com
```

Пример конфигурации

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - pool.ntp.org
      - ntp.ubuntu.com
      - time.google.com
  version: 1
```

FAQ

Как запретить использование chrony и использовать NTP-демоны на узлах?

1. Выключите модуль chrony.
2. Создайте `NodeGroupConfiguration` custom step, чтобы включить NTP-демоны на узлах (пример для `systemd-timesyncd`):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: enable-ntp-on-node.sh
spec:
  weight: 100
  nodeGroups: ["*"]
  bundles: ["*"]
  content: |
    systemctl enable systemd-timesyncd
    systemctl start systemd-timesyncd
```

Модуль cilium-hubble



Веб-интерфейсы, связанные с модулем: hubble

Модуль `cilium-hubble` обеспечивает визуализацию сетевого стека кластера, если включен Cilium CNI.

Требования

Для работы модуля `cilium-hubble` необходимы:

- Версия ядра Linux ≥ 5.8 с поддержкой eBPF.
- Поддержка формата метаданных BPF Type Format (BTF). Проверить можно следующими способами:
 - выполнить команду `ls -lah /sys/kernel/btf/vmlinux` — наличие файла подтверждает поддержку BTF;
 - выполнить команду `grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*` — если параметр включён, BTF поддерживается.

Настройки

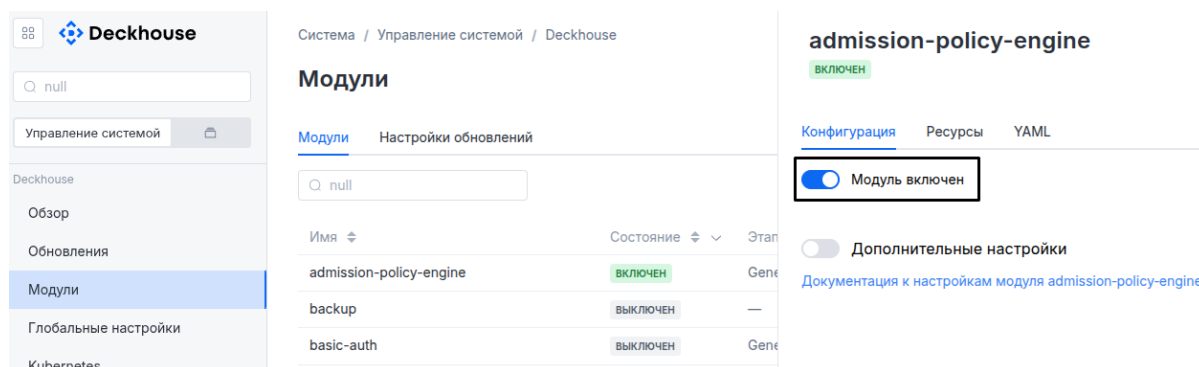
Модуль не включен по умолчанию в каком-либо наборе модулей.

Если модуль `spi-cilium` выключен, параметр `ciliumHubbleEnabled` не повлияет на включение модуля `cilium-hubble`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cilium-hubble`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cilium-hubble`:

```
d8 system module enable cilium-hubble
```

- **С помощью ModuleConfig `cilium-hubble`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cilium-hubble` (создайте его, при необходимости).

Пример манифеста для включения модуля `cilium-hubble`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cilium-hubble
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cilium-hubble`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

[Конфигурация](#) [Ресурсы](#) [YAML](#)

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

[Конфигурация](#) [YAML](#)

Вы также можете отредактировать объект ModuleConfig `cilium-hubble` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cilium-hubble`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cilium-hubble` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cilium-hubble
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cilium-hubble`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cilium-hubble`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cilium-hubble
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `cni-cilium`: любая версия.

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Описание конверсии отсутствует.

Преобразования, выполняемые конверсией (синтаксис jq)...

- `del(.auth.password) | if .auth == {} then del(.auth) end`

Параметры

Версия схемы: 2

- **settings**

объект

- **settings.auth**

объект

Опции, связанные с аутентификацией и авторизацией доступа к веб-интерфейсу Hubble.

- **settings.auth.allowedUserEmails**

массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.allowedUserGroups**

массив строк

Список групп, пользователям которых разрешен доступ к веб-интерфейсу Hubble.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.externalAuthentication**

объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX external-auth, работающий на основе модуля Nginx auth_request).

i

Внешняя аутентификация включается автоматически, если включен модуль `user-authn`.

- **settings.auth.externalAuthentication.authSignInURL**

строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

Пример:

```
authSignInURL: https://example.com/dex/sign_in
```

- **settings.auth.externalAuthentication.authURL**

строка

URL сервиса аутентификации.

Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

Пример:

```
authURL: https://example.com/dex/auth
```

- **settings.auth.whitelistSourceRanges**

массив строк

Список адресов (CIDR), которым разрешен доступ к веб-интерфейсу Hubble.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **settings.debugLogging**

булевый

Включает отладочный уровень логирования для компонентов Hubble.

По умолчанию: `false`

- **settings.https**

объект

Тип используемого сертификата.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.https.certManager**

объект

Настройки для certmanager.

■ **settings.https.certManager.clusterIssuerName**

строка

Тип ClusterIssuer, используемого для заказа SSL-сертификата. В данный момент доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

Примеры:

```
clusterIssuerName: letsencrypt
```

```
clusterIssuerName: letsencrypt-staging
```

```
clusterIssuerName: selfsigned
```

■ **settings.https.customCertificate**

объект

Настройки для использования пользовательского сертификата.

■ **settings.https.customCertificate.secretName**

строка

Имя Secret в пространстве имен `d8-system`, который будет использоваться для веб-интерфейса Hubble.

Secret должен быть в формате `kubernetes.io/tls`.

■ **settings.https.mode**

строка

Режимы работы HTTPS:

- `CertManager` — доступ по HTTPS с заказом сертификата согласно ClusterIssuer, заданному в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — доступ по HTTPS с использованием сертификата из пространства имен `d8-system`.
- `Disabled` — доступ только по HTTP.
- `OnlyInURI` — доступ по HTTP с внешним HTTPS-балансировщиком. Балансировщик терминирует HTTPS, и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `CertManager`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для Hubble.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, то Deckhouse попытается вычислить значение `nodeSelector` автоматически.

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, то Deckhouse попытается вычислить значение `tolerations` автоматически.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Аутентификация

По умолчанию используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication`. Если эти варианты отключены, модуль включит базовую аутентификацию со сгенерированным паролем.

Чтобы просмотреть сгенерированный пароль, выполните команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values cilium-hubble -O json | jq '.ciliumHubble.internal.auth.password'
```

Чтобы сгенерировать новый пароль, удалите Secret:

```
d8 k -n d8-cni-cilium delete secret/hubble-basic-auth
```



Параметр `auth.password` больше не поддерживается.

Cloud provider — AWS

Модуль `cloud-provider-aws` обеспечивает интеграцию Deckhouse Kubernetes Platform с Amazon AWS. Он предоставляет возможность модулю `node-manager` использовать ресурсы AWS при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-aws`:

- Управление ресурсами AWS через `cloud-controller-manager`:
 - создаёт сетевые маршруты для сети `PodNetwork` на стороне AWS;
 - создаёт балансировщики нагрузки для Service с типом `LoadBalancer`;
 - актуализирует метаданные узлов кластера и удаляет из Kubernetes узлы, которых больше нет в AWS.
- Заказ дисков через CSI-драйвер EBS (`ebs.csi.aws.com`) и создание `StorageClass` для типов томов AWS, чтобы из кластера можно было заказывать `PersistentVolume`.
- Заказ базовой инфраструктуры и `CloudPermanent`-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-aws`.
- Заказ `CloudEphemeral`-узлов через Machine Controller Manager (MCM). Параметры виртуальных машин задаются в ресурсе `AWSInstanceClass`.
- Регистрация в модуле `node-manager`, чтобы `AWSInstanceClass` можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.

Настройки

В большинстве случаев нет необходимости в ручной конфигурации модуля. Модуль настраивается автоматически, исходя из выбранной схемы размещения (custom resource `AWSClusterConfiguration`).

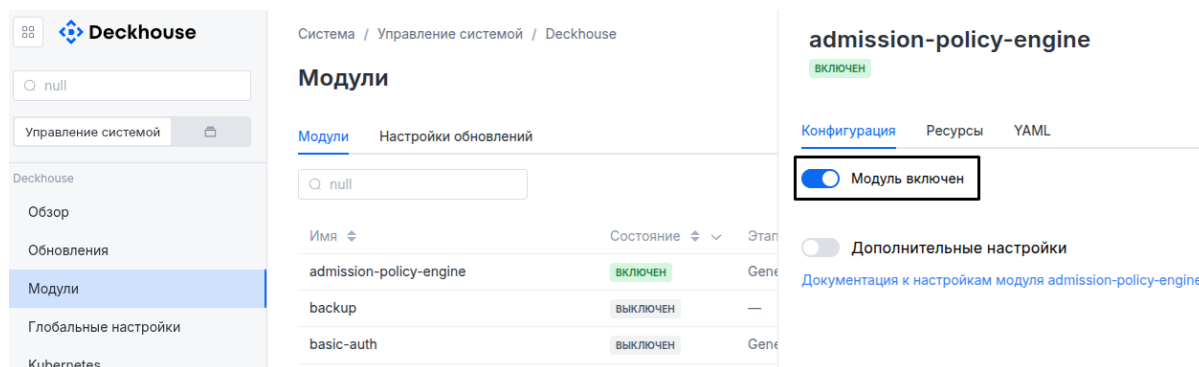
Количество и параметры процесса заказа машин в облаке настраиваются в custom resource `NodeGroup` модуля `node-manager`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference NodeGroup`). Инстанс-класс для cloud-провайдера AWS — это custom resource `AWSInstanceClass`, в котором указываются конкретные параметры самих машин.

Модуль автоматически создает `StorageClasses`, которые есть в AWS: `gp3`, `gp2`, `sc1` и `st1`, позволяет сконфигурировать диски с необходимым IOPS, а также отфильтровать ненужные `StorageClass`, указав их в параметре `exclude`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-aws`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-aws`:

```
d8 system module enable cloud-provider-aws
```

- **С помощью ModuleConfig `cloud-provider-aws`.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig cloud-provider-aws` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-aws`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-aws
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-aws`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cloud-provider-aws` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-aws`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-aws` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-aws
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-aws`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-aws`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-aws
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.storageClass**
объект
 - **settings.storageClass.default**
Параметр устарел строка

Имя StorageClass, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- присутствующий StorageClass в кластере, имеющий default-аннотацию;
- первый (по алфавиту) StorageClass из тех, что создаются модулем.

i **Параметр устарел.** Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`

Пример:

```
default: gp3
```

■ **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) StorageClass, которые не будут созданы в кластере.

Пример:

```
exclude:
- sc.*
- st1
```

■ **settings.storageClass.provision**

массив объектов

Создание дополнительных StorageClass или переопределение параметров, создаваемых по умолчанию.

■ **settings.storageClass.provision.iops**

строка

I/O операций в секунду для дисков типа `gp3`:

- Допустимые значения: от `3000` до `16000`.
- Если не задан, будет использоваться значение `3000` согласно документации.
- Изменение этого параметра не приводит к изменению аналогичных параметров для уже созданных дисков. Их необходимо править вручную через консоль управления AWS.

■ **settings.storageClass.provision.iopsPerGB**

строка

I/O операций в секунду на каждый Гб для дисков типа `io1` или `io2` (у дисков `gp2` этот параметр `3`):

- **Внимание!** Если это значение, умноженное на размер запрошенного диска, будет меньше `100` или больше `64000`, создание такого диска завершится ошибкой.
- Подробное описание типов дисков и их IOPS вы найдете в официальной документации.

■ **settings.storageClass.provision.name**

строка

Обязательный параметр

Имя будущего класса.

■ **settings.storageClass.provision.throughput**

строка

Пропускная способность в MiB/s для дисков типа `gp3`:

- Допустимые значения: от `125` до `1000`.
- Если не задан, будет использоваться значение `125` согласно документации.
- Изменение этого параметра не приводит к изменению аналогичных параметров для уже созданных дисков. Их необходимо править вручную через консоль управления AWS.

■ **settings.storageClass.provision.type**

строка

Обязательный параметр

Тип диска.

Допустимые значения: `gp3`, `gp2`, `sc1`, `st1`, `io1`, `io2`

Настройки провайдера

AWSClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в AWS.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 172.16.0.0/22
vpcNetworkCIDR: 172.16.0.0/16
masterNodeGroup:
  replicas: 1
  instanceClass:
    instanceType: m5.xlarge
    ami: ami-08b6d44b4f6f7b279
    diskType: gp3
nodeGroups:
- name: worker
  nodeTemplate:
    labels:
      node-role.kubernetes.io/worker: ''
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aada10862ade98f27
    additionalTags:
      backup: srv1
  provider:
    providerAccessKeyId: "<AWS_ACCESS_KEY>"
    providerSecretAccessKey: "<AWS_SECRET_ACCESS_KEY>"
    region: eu-central-1
  tags:
    team: rangers
```

- **additionalRolePolicies**

массив строк

Список дополнительных policy actions для IAM-ролей.

- Дополнительные policy actions будут добавлены к базовым политикам IAM-ролей.
- Необязательный параметр. Если отсутствует, IAM-роли будут использовать только policy actions по умолчанию.
- Пример policy actions: `ecr:ListImages`, `s3:GetObject` etc

По умолчанию IAM-роли содержат следующие policy actions:

- `ec2:DescribeTags`
- `ec2:DescribeInstances`
- **Элемент массива** строка

Шаблон: `^[a-zA-Z0-9_-]+:[a-zA-Z0-9_-]*$`

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **disableDefaultSecurityGroup**

булевый

Если установлено значение `true`, то security group по умолчанию создаваться не будут.

Важно. При использовании `disableDefaultSecurityGroup: true` вы обязаны самостоятельно создать все необходимые security group для доступа к узлам кластера. Кроме того, необходимо явно указать их в следующих параметрах:

- `additionalSecurityGroups` в секции `masterNodeGroup` песцыса `AWSClusterConfiguration`;
- `additionalSecurityGroups` в песцысе `AWSInstanceClass`;
- `additionalSecurityGroups` в секции `nodeGroups.instanceClass`.

Для настройки security group, используемых балансировщиками нагрузки, укажите их через аннотацию `service.beta.kubernetes.io/aws-load-balancer-security-groups`.

• **existingVPCID**

строка

ID существующего VPC, в котором будет развернута схема.

- **Обязательный параметр**, если не указан `vpcNetworkCIDR`.
- **Важно!** Если в данной VPC уже есть Internet Gateway, деплой базовой инфраструктуры упадет с ошибкой. На данный момент использовать уже существующий Internet Gateway нельзя.

• **iamNodeRole**

строка

Имя IAM-роли, которая будет привязана ко всем AWS-инстансам узлов кластера.

DKP всегда создает и привязывает к каждому AWS-инстансу узла кластера специальную IAM-роль с именем `<PREFIX>-node`, где `<PREFIX>` — значение параметра `cloud.prefix` общих параметров кластера. Вы можете указать в параметре `iamNodeRole` свою IAM-роль с большими правами, но, важно чтобы она включала в себя политики IAM-роли, создаваемой DKP (роль `<PREFIX>-node`).

Подробнее о IAM-ролях для AWS EC2 можно прочитать в документации AWS.

Шаблон: `^[a-zA-Z0-9+=[,.\@\_']{1,64}$`

• **kind**

строка

Обязательный параметр

Допустимые значения: `AWSClusterConfiguration`

• **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

i

Схема размещения `Standard` считается **устаревшей**, не используйте ее.

Допустимые значения: `WithoutNAT`, `WithNAT`, `Standard`

• **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.

i

Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.additionalTags**
объект

Дополнительные теги, которые будут присвоены созданным инстансам в дополнение к указанным в конфигурации cloud-провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

- **masterNodeGroup.instanceClass**
объект

Обязательный параметр

Частичное содержимое полей `AWSInstanceClass`.

- **masterNodeGroup.instanceClass.additionalSecurityGroups**

массив строк

Дополнительные security groups, которые будут присвоены созданным инстансам.

- **masterNodeGroup.instanceClass.ami**

строка

Обязательный параметр

Образ (AMI ID), который будет использоваться в заказанных инстансах.

Как найти нужный AMI (в каждом регионе AMI разные):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-
server-2020*' | \
jq '.Images[].ImageId'
```

Пример:

```
ami: ami-040a1551f9c9d11ad
```

- **masterNodeGroup.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в **Гиб**.

Пример:

```
diskSizeGb: 50
```

- **masterNodeGroup.instanceClass.diskType**

строка

Тип созданного диска.

Допустимые значения: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

Пример:

```
diskType: gp2
```

- **masterNodeGroup.instanceClass.etcdDisk**

объект

По умолчанию: `{"sizeGb":20,"type":"gp3"}`

- **masterNodeGroup.instanceClass.etcdDisk.sizeGb**

целочисленный

Размер диска для etcd. Значение указывается в **Гиб**.

- **masterNodeGroup.instanceClass.etcdDisk.type**

строка

Тип диска для etcd.

- **masterNodeGroup.instanceClass.instanceType**

строка

Обязательный параметр

Тип заказываемых инстансов.

i **Внимание.** Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

Пример:

```
instanceType: t3.large
```

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **masterNodeGroup.zones**

массив строк

Необязательный параметр.

Список зон, в которых допустимо создавать master-узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

- **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.additionalTags**

объект

Дополнительные к основным (`AWSClusterConfiguration.tags`) теги, которые будут присвоены созданным экземплярам.

Пример:

```
project: cms-production
severity: critical
```

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `AWSInstanceClass`.

- **nodeGroups.instanceClass.additionalSecurityGroups**

массив строк

Дополнительные security groups, которые будут присвоены созданным экземплярам.

- **nodeGroups.instanceClass.ami**

строка

Обязательный параметр

Образ (AMI ID), который будет использоваться в заказанных экземплярах.

Как найти нужный AMI (в каждом регионе AMI разные):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-
server-2020*' | \
jq '.Images[].ImageId'
```

Пример:

```
ami: ami-040a1551f9c9d11ad
```

- **nodeGroups.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

Пример:

```
diskSizeGb: 50
```

- **nodeGroups.instanceClass.diskType**

строка

Тип созданного диска.

Допустимые значения: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

Пример:

```
diskType: gp2
```

■ **nodeGroups.instanceClass.instanceType**

строка

Обязательный параметр

Тип заказываемых инстансов.

i **Внимание.** Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

Пример:

```
instanceType: t3.large
```

○ **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup. Используется для генерации имен узлов.

○ **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узла.

■ **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: "true"
```

■ **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

■ **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```


- **publicNetworkAllowList**

массив строк

Список CIDR-адресов, которым разрешён доступ к публичным эндпоинтам кластера.

Важно. AWS ограничивает количество входящих и исходящих правил в одной группе безопасности (security group) до 60. Если требуется указать больше CIDR-адресов, запросите увеличение квоты через поддержку AWS.

По умолчанию: `["0.0.0.0/0"]`

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **sshAllowList**

массив строк

Список CIDR, разрешенных для подключения к узлам по SSH.

По умолчанию, `0.0.0.0/0`.

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **standard**

Параметр устарел
объект

Схема размещения `Standard` считается **устаревшей**, не используйте ее.

- **tags**

объект

Необязательный параметр.

Словарь тегов, которые будут созданы на всех ресурсах, имеющих такую возможность.

Если поменять теги в рабочем кластере, после применения изменений необходимо пересоздать все машины.

- **vpcNetworkCIDR**

строка

Подсеть, которая будет указана в созданном VPC.

Обязательный параметр, если не указан параметр для развертывания в уже созданном VPC `existingVPCID`.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withNAT**

объект

- **withNAT.bastionInstance**

объект

- **withNAT.bastionInstance.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `AWSInstanceClass`.

- **withNAT.bastionInstance.instanceClass.additionalSecurityGroups**

массив строк

Дополнительные security groups, которые будут присвоены созданному экземпляру.

- **withNAT.bastionInstance.instanceClass.ami**

строка

Обязательный параметр

Образ (AMI ID), который будет использоваться для создания заказываемого экземпляра.

Как найти нужный AMI (в каждом регионе AMI разные):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-
amd64-server-2020*' | \
jq '.Images[].ImageId'
```

Пример:

```
ami: ami-040a1551f9c9d11ad
```

■ **withNAT.bastionInstance.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

Пример:

```
diskSizeGb: 20
```

■ **withNAT.bastionInstance.instanceClass.diskType**

строка

Тип созданного root-диска.

Допустимые значения: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

Пример:

```
diskType: gp2
```

■ **withNAT.bastionInstance.instanceClass.instanceType**

строка

Обязательный параметр

Тип заказываемого инстанса.

Пример:

```
instanceType: t3.large
```

■ **withNAT.bastionInstance.zone**

строка

Зона, в которой будет создан инстанс bastion.

По умолчанию будет использоваться первая доступная зона в регионе или первая из списка глобального параметра `zones`.

• **withoutNAT**

объект

• **zones**

массив строк

Глобальное ограничение набора зон, с которыми работает cloud provider.

Ограничивает только **создание** узлов в кластере.

Подготовка окружения



Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.

В Amazon AWS используются дистрибутивы Ubuntu с кастомным ядром без `erofs`. Чтобы привести ОС в соответствие с требованиями и добавить необходимые модули, можно использовать команду `sudo apt install linux-modules-extra-aws`.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

Для работы `cloud-provider` и `machine-controller-manager` требуется доступ в API AWS из-под IAM-пользователя, который обладает достаточным набором прав.



Убедитесь в наличии доступа к нужному региону и наличии необходимых квот.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

JSON-спецификация Policy

Сначала подготовьте JSON-файл с конфигурацией необходимых прав:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "ec2:AllocateAddress",
        "ec2:AssociateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AttachVolume",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateKeyPair",
        "ec2:CreateNATGateway",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:CreateVpc",
        "ec2>DeleteInternetGateway",
        "ec2>DeleteKeyPair",
        "ec2>DeleteNATGateway",
        "ec2>DeleteRoute",
        "ec2>DeleteRouteTable",
        "ec2>DeleteSecurityGroup",
        "ec2>DeleteSubnet",
        "ec2>DeleteTags",
        "ec2>DeleteVolume",
        "ec2>DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAddressesAttribute",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeImages",
        "ec2:DescribeInstanceAttribute",
        "ec2:DescribeInstanceCreditSpecifications",
        "ec2:DescribeInstances",
```

```

"ec2:DescribeInstanceTypes",
"ec2:DescribeInstanceTopology",
"ec2:DescribeInternetGateways",
"ec2:DescribeKeyPairs",
"ec2:DescribeNatGateways",
"ec2:DescribeNetworkInterfaces",
"ec2:DescribeRegions",
"ec2:DescribeRouteTables",
"ec2:DescribeSecurityGroups",
"ec2:DescribeSecurityGroupRules",
"ec2:DescribeSubnets",
"ec2:DescribeTags",
"ec2:DescribeVolumesModifications",
"ec2:DescribeVolumes",
"ec2:DescribeVpcAttribute",
"ec2:DescribeVpcClassicLink",
"ec2:DescribeVpcClassicLinkDnsSupport",
"ec2:DescribeVpcs",
"ec2:DetachInternetGateway",
"ec2:DetachVolume",
"ec2:DisassociateAddress",
"ec2:DisassociateRouteTable",
"ec2:ImportKeyPair",
"ec2:ModifyInstanceAttribute",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVolume",
"ec2:ModifyVpcAttribute",
"ec2:ReleaseAddress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress",
"ec2:RunInstances",
"ec2:TerminateInstances",
"ec2:DescribeVpcPeeringConnections",
"ec2:CreateVpcPeeringConnection",
"ec2>DeleteVpcPeeringConnection",
"ec2:AcceptVpcPeeringConnection",
"ec2:CreateNetworkInterface",
"ec2:DescribeNetworkInterfaceAttribute",
"ec2:ModifyNetworkInterfaceAttribute",
"ec2>DeleteNetworkInterface",
"ec2:DescribeNetworkInterfaces",
"elasticloadbalancing:AddTags",
"elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
"elasticloadbalancing:AttachLoadBalancerToSubnets",
"elasticloadbalancing:ConfigureHealthCheck",
"elasticloadbalancing:CreateListener",
"elasticloadbalancing:CreateLoadBalancer",
"elasticloadbalancing:CreateLoadBalancerListeners",
"elasticloadbalancing:CreateLoadBalancerPolicy",
"elasticloadbalancing:CreateTargetGroup",
"elasticloadbalancing>DeleteListener",
"elasticloadbalancing>DeleteLoadBalancer",
"elasticloadbalancing>DeleteLoadBalancerListeners",
"elasticloadbalancing>DeleteTargetGroup",
"elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
"elasticloadbalancing:DeregisterTargets",
"elasticloadbalancing:DescribeListeners",
"elasticloadbalancing:DescribeLoadBalancerAttributes",
"elasticloadbalancing:DescribeLoadBalancerPolicies",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeTargetGroups",
"elasticloadbalancing:DescribeTargetHealth",
"elasticloadbalancing:DetachLoadBalancerFromSubnets",
"elasticloadbalancing:ModifyListener",
"elasticloadbalancing:ModifyLoadBalancerAttributes",
"elasticloadbalancing:ModifyTargetGroup",
"elasticloadbalancing:RegisterInstancesWithLoadBalancer",

```



```

        "elasticloadbalancing:RegisterTargets",
        "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
        "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
        "iam:AddRoleToInstanceProfile",
        "iam:CreateInstanceProfile",
        "iam:CreateRole",
        "iam:CreateServiceLinkedRole",
        "iam:DeleteInstanceProfile",
        "iam:DeleteRole",
        "iam:DeleteRolePolicy",
        "iam:GetInstanceProfile",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "iam:ListInstanceProfilesForRole",
        "iam:ListRolePolicies",
        "iam:ListAttachedRolePolicies",
        "iam:PassRole",
        "iam:PutRolePolicy",
        "iam:RemoveRoleFromInstanceProfile",
        "iam:TagRole",
        "kms:DescribeKey",
        "sts:GetCallerIdentity"
    ],
    "Resource": "*"
}
]
}

```

Инструкции, как применить этот JSON-файл, ниже.

Настройка IAM через веб-интерфейс

Чтобы настроить IAM через веб-интерфейс, сначала создайте новую политику (Policy) и примените к ней созданный ранее JSON-файл:

1. Откройте Identity and Access Management (IAM).
2. Перейдите в раздел Policies и нажмите Create Policy.
3. Выберите вкладку JSON и вставьте приведенную выше спецификацию.
4. Нажмите Next: Tags, затем Next: Review.
5. Задайте название политики в поле Name (например, D8CloudProviderAWS).
6. Нажмите Create Policy.

Затем добавьте нового пользователя:

1. Перейдите в раздел Users IAM и нажмите Add users.
2. Задайте имя в поле User name (например, deckhouse).

Примените к нему созданную Policy:

1. В разделе Select AWS credential type выберите Access key - Programmatic access.
2. Нажмите Next: Permissions.
3. Выберите вкладку Attach existing policies directly.
4. Введите в поле поиска (Filter policies) имя политики, указанное на предыдущих шагах (например, D8CloudProviderAWS), и в полученном списке отметьте checkbox напротив искомой политики.
5. Нажмите Next: Tags, затем Next: Review.
6. Нажмите Create user.

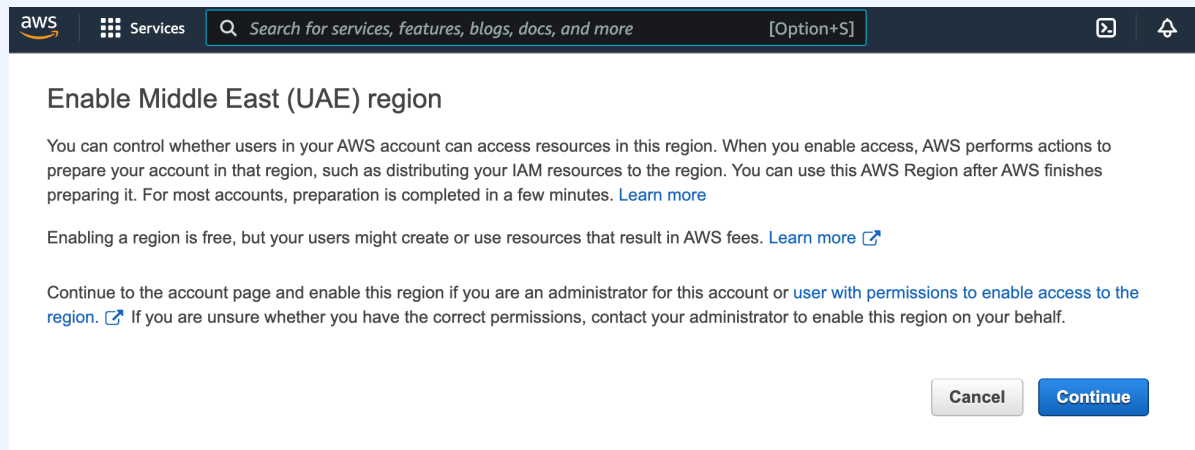
Сохраните полученные Access key ID и Secret access key.



Проверьте, есть ли у вашей учетной записи (и, соответственно, у созданного пользователя) доступ к нужным регионам. Для этого выберите необходимый регион в выпадающем списке в правом верхнем углу.

Если произойдет переключение в выбранный регион, доступ к региону есть.

Если доступа к региону нет, вы получите следующее сообщение (может отличаться):



В этом случае нажмите **Continue**, чтобы разрешить использование региона.

Настройка IAM через CLI

Также IAM можно настроить через интерфейс командной строки.

Для этого с помощью следующей команды сохраните JSON-спецификацию в файл `policy.json`:

```
cat > policy.json << EOF
<Policy JSON spec>
EOF
```

Затем создайте новую Policy с именем `D8CloudProviderAWS` и примечанием `ARN`, используя JSON-спецификацию из файла `policy.json`:

```
aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
```

В ответ отобразится следующий текст:

```
{
  "Policy": {
    "PolicyName": "D8CloudProviderAWS",
    "PolicyId": "AAA",
    "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2020-08-27T02:52:06+00:00",
    "UpdateDate": "2020-08-27T02:52:06+00:00"
  }
}
```

Создайте нового пользователя:

```
aws iam create-user --user-name deckhouse
```

В ответ отобразится следующий текст:

```
{
  "User": {
    "Path": "/",
    "UserName": "deckhouse",
    "UserId": "AAAXXX",
    "Arn": "arn:aws:iam::123:user/deckhouse",
    "CreateDate": "2020-08-27T03:05:42+00:00"
  }
}
```

Разрешите доступ к API и сохраните пару `AccessKeyId` + `SecretAccessKey`:

```
aws iam create-access-key --user-name deckhouse
```

В ответ отобразится следующий текст:

```
{
  "AccessKey": {
    "UserName": "deckhouse",
    "AccessKeyId": "XXYYYY",
    "Status": "Active",
    "SecretAccessKey": "ZZZZzz",
    "CreateDate": "2020-08-27T03:06:22+00:00"
  }
}
```

Объедините `User` и `Policy`:

```
aws iam attach-user-policy --user-name username --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
```

Настройка IAM через Terraform

Пример настройки IAM через Terraform:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name       = "D8CloudProviderAWS"
  path       = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON-спецификация Policy>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user          = aws_iam_user.user.name
  policy_arn    = aws_iam_policy.policy.arn
}
```

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

AWSInstanceClass

Scope: Cluster

- v1
- v1alpha1

Параметры инстанса в AWS, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalSecurityGroups**
массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего InstanceClass.

- **spec.additionalTags**
объект

Дополнительные теги, которые будут присвоены созданным инстансам.

- **spec.ami**
строка

Образ Amazon Machine Image (AMI ID), который будет использоваться в заказанных инстансах.

Если он не указан, используется AMI, определенный в параметре `masterNodeGroup.instanceClass.ami` ресурса `AWSClusterConfiguration`.

Как найти нужный AMI (в каждом регионе AMI разные):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
jq '.Images[].ImageId'
```

Пример:

```
ami: ami-040a1551f9c9d11ad
```

- **spec.capacity**
Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `instanceType`).

Используется `cluster-autoscaler`’ом при планировании, только когда в `NodeGroup`’е еще нет узлов (при `minPerZone` равном 0). Если в `NodeGroup` уже есть узлы, `cluster-autoscaler` использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`t2.micro`, `m5a.12xlarge` и т. п.) заложены в Deckhouse.

- **spec.capacity.cpu**
строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

- **spec.capacity.memory**
строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.diskSizeGb**
целочисленный

Размер root-диска. Значение указывается в Гиб.

По умолчанию: 20

Пример:

```
diskSizeGb: 20
```

- **spec.diskType**
строка

Тип созданного диска.

По умолчанию: gp2

Пример:

```
diskType: gp2
```

- **spec.instanceType**
строка

Обязательный параметр

Тип заказываемых инстансов.

Внимание! Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

Пример:

```
instanceType: t3.large
```

- **spec.iops**
строка

Количество IOPS. Применяется только для `diskType io1`.

Пример:

```
iops: '500'
```

- **spec.spot**
булевый

Создание spot-инстансов (spot instance). Spot-инстансы будут запускаться с минимально возможной для успешного запуска ценой за час.

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

Параметры инстанса в AWS, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalSecurityGroups**
массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего InstanceClass.

- **spec.additionalTags**
объект

Дополнительные теги, которые будут присвоены созданным инстансам.

- **spec.ami**
строка

Образ Amazon Machine Image (AMI ID), который будет использоваться в заказанных инстансах.

Если он не указан, используется AMI, определенный в параметре `masterNodeGroup.instanceClass.ami` ресурса `AWSClusterConfiguration`.

Как найти нужный AMI (в каждом регионе AMI разные):

```
aws ec2 --region <REGION> describe-images \
  --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
  jq '.Images[].ImageId'
```

Пример:

```
ami: ami-040a1551f9c9d11ad
```

- **spec.capacity**
Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `instanceType`).

Используется `cluster-autoscaler`ом при планировании, только когда в `NodeGroup`’е еще нет узлов (при `minPerZone` равном 0). Если в `NodeGroup` уже есть узлы, `cluster-autoscaler` использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`t2.micro`, `m5a.12xlarge` и т. п.) заложены в Deckhouse.

- **spec.capacity.cpu**
строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

- **spec.capacity.memory**
строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.diskSizeGb**
целочисленный

Размер root-диска. Значение указывается в Гиб.

По умолчанию: 20

Пример:

```
diskSizeGb: 20
```

- **spec.diskType**
строка

Тип созданного диска.

По умолчанию: `gp2`

Пример:

```
diskType: gp2
```

- **spec.instanceType**

строка

Обязательный параметр

Тип заказываемых инстансов.

Внимание! Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

Пример:

```
instanceType: t3.large
```

- **spec.iops**

строка

Количество IOPS. Применяется только для `diskType io1`.

Пример:

```
iops: '500'
```

- **spec.spot**

булевый

Создание spot-инстансов (spot instance). Spot-инстансы будут запускаться с минимально возможной для успешного запуска ценой за час.

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

FAQ

Как поднять пиринговое соединение между VPC?

Для примера рассмотрим настройку пирингового соединения между двумя условными VPC — vpc-a и vpc-b.

i

Важно! IPv4 CIDR у обоих VPC должен различаться.

Для настройки выполните следующие шаги:

1. Перейдите в регион, где работает vpc-a.
2. Нажмите VPC -> VPC Peering Connections -> Create Peering Connection и настройте пиринговое соединение:
 - Name: vpc-a-vpc-b.
 - Заполните Local и Another VPC.
3. Перейдите в регион, где работает vpc-b.
4. Нажмите VPC -> VPC Peering Connections.
5. Выделите созданное соединение и выберите Action "Accept Request".
6. Для vpc-a добавьте во все таблицы маршрутизации маршруты до CIDR vpc-b через пиринговое соединение.
7. Для vpc-b добавьте во все таблицы маршрутизации маршруты до CIDR vpc-a через пиринговое соединение.

Как создать кластер в новом VPC с доступом через имеющийся bastion-хост?

1. Выполнить bootstrap базовой инфраструктуры кластера:

```
dhctl bootstrap-phase base-infra --config config
```

2. Поднять пиринговое соединение по инструкции выше.
3. Продолжить установку кластера. На вопрос про кеш Terraform ответить `y`:

```
dhctl bootstrap --config config --ssh-...
```

Как создать кластер в новом VPC и развернуть bastion-хост для доступа к узлам?

1. Выполнить bootstrap базовой инфраструктуры кластера:

```
dhctl bootstrap-phase base-infra --config config
```

2. Запустить вручную bastion-хост в subnet -public-0.
3. Продолжить установку кластера. На вопрос про кеш Terraform ответить `y`:

```
dhctl bootstrap --config config --ssh-...
```

Особенности настройки bastion

Поддерживаются сценарии:

- bastion-хост уже создан во внешней VPC:
 1. Создайте базовую инфраструктуру кластера — `dhctl bootstrap-phase base-infra`.
 2. Настройте пиринговое соединение между внешней и свежесозданной VPC.
 3. Продолжите установку с указанием bastion-хоста — `dhctl bootstrap --ssh-bastion...`
- bastion-хост требуется поставить в свежесозданной VPC:
 1. Создайте базовую инфраструктуру кластера — `dhctl bootstrap-phase base-infra`.
 2. Запустите вручную bastion-хост в subnet -public-0.
 3. Продолжите установку с указанием bastion-хоста — `dhctl bootstrap --ssh-bastion...`

Добавление CloudStatic-узлов в кластер

Для добавления виртуальной машины в качестве узла в кластер выполните следующие шаги:

1. Прикрепите группу безопасности `<prefix>-node` к виртуальной машине.
2. Прикрепите IAM-роль `<prefix>-node` к виртуальной машине.
3. Укажите следующие теги у виртуальной машины (чтобы `cloud-controller-manager` смог найти виртуальные машины в облаке):

```
"kubernetes.io/cluster/<cluster_uuid>" = "shared"
"kubernetes.io/cluster/<prefix>" = "shared"
```

- Узнать `cluster_uuid` можно с помощью команды:

```
d8 k -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
```

- Узнать `prefix` можно с помощью команды:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
| base64 -d | grep prefix
```

Как увеличить размер volume в кластере?

Задайте новый размер в соответствующем ресурсе PersistentVolumeClaim в параметре `spec.resources.requests.storage`.

Операция проходит полностью автоматически и занимает до одной минуты. Никаких дополнительных действий не требуется.

За ходом процесса можно наблюдать в events через команду `d8 k describe pvc`.

i

После изменения volume нужно подождать не менее шести часов и убедиться, что volume находится в состоянии `in-use` или `available`, прежде чем станет возможно изменить его еще раз. Подробности можно найти в официальной документации.

Как настроить доступ к репозиторию Amazon ECR на узлах кластера

i

Под репозиторием Amazon ECR подразумевается Amazon ECR repository.

1. Задайте права для чтения образов в Repository policies. В `Principal` должен быть существующий объект `Roles`.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RepositoryRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::xxx:role/xxx-node"
      },
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:DescribeRepositories",
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:ListImages",
        "ecr:ListTagsForResource"
      ]
    }
  ]
}
```

Примените эту политику в `Amazon ECR > Private registry > Repositories > `` > Permissions`.

2. Добавьте `ecr:GetAuthorizationToken` в `additionalRolePolicies`.

Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?

Если переключение на заказ узлов в менее приоритетных группах занимает много времени (например, когда для групп узлов со spot-инстансами установлен наивысший приоритет и при недоступности таких инстансов заказ узлов из других групп происходит очень долго), воспользуйтесь инструкцией.

Примеры

Пример custom resource `AWSInstanceClass`

Ниже представлен простой пример конфигурации custom resource `AWSInstanceClass`:

```
apiVersion: deckhouse.io/v1
kind: AWSInstanceClass
metadata:
  name: worker
spec:
  instanceType: t3.large
  ami: ami-040a1551f9c9d11ad
  diskSizeGb: 15
  diskType: gp2
```

LoadBalancer

Аннотации объекта `Service`

Поддерживаются следующие параметры в дополнение к существующим в `upstream`:

1. `service.beta.kubernetes.io/aws-load-balancer-type` — может иметь значение `none`, что приведет к созданию **только** целевой группы, без какого-либо `LoadBalancer`'а.
2. `service.beta.kubernetes.io/aws-load-balancer-backend-protocol` — используется в связке с `service.beta.kubernetes.io/aws-load-balancer-type: none`:
 - Возможные значения:
 - `tcp` (по умолчанию);
 - `tls`;
 - `http`;
 - `https`.
 - **Внимание!** При изменении этого параметра `cloud-controller-manager` попытается пересоздать целевую группу. Если к ней уже привязаны NLB или ALB, удалить целевую группу не получится и он будет бесконечно пытаться это сделать. В таком случае необходимо вручную отсоединить NLB или ALB от целевой группы.

Настройка политик безопасности на узлах

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера в AWS, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим виртуальным машинам в облаке по требованию службы безопасности.

Для всех них следует применять дополнительные группы безопасности (security group). Можно использовать только предварительно созданные в облаке группы безопасности.

Установка дополнительных security groups на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности (security group) указываются в `AWSClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей соответствующую `nodeGroup`, в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` содержит массив строк с именами групп безопасности.

Установка дополнительных security groups на эфемерных узлах

Необходимо указать параметр `additionalSecurityGroups` для всех `AWSInstanceClass` в кластере, которым нужны дополнительные группы безопасности (security group).

Настройка балансировщика в случае наличия Ingress-узлов не во всех зонах

Необходимо указать аннотацию на объекте Service: `service.beta.kubernetes.io/aws-load-balancer-subnets: subnet-foo, subnet-bar`.

Чтобы получить список текущих подсетей, используемых для конкретной установки, выполните следующую команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values cloud-provider-aws -O json \
| jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

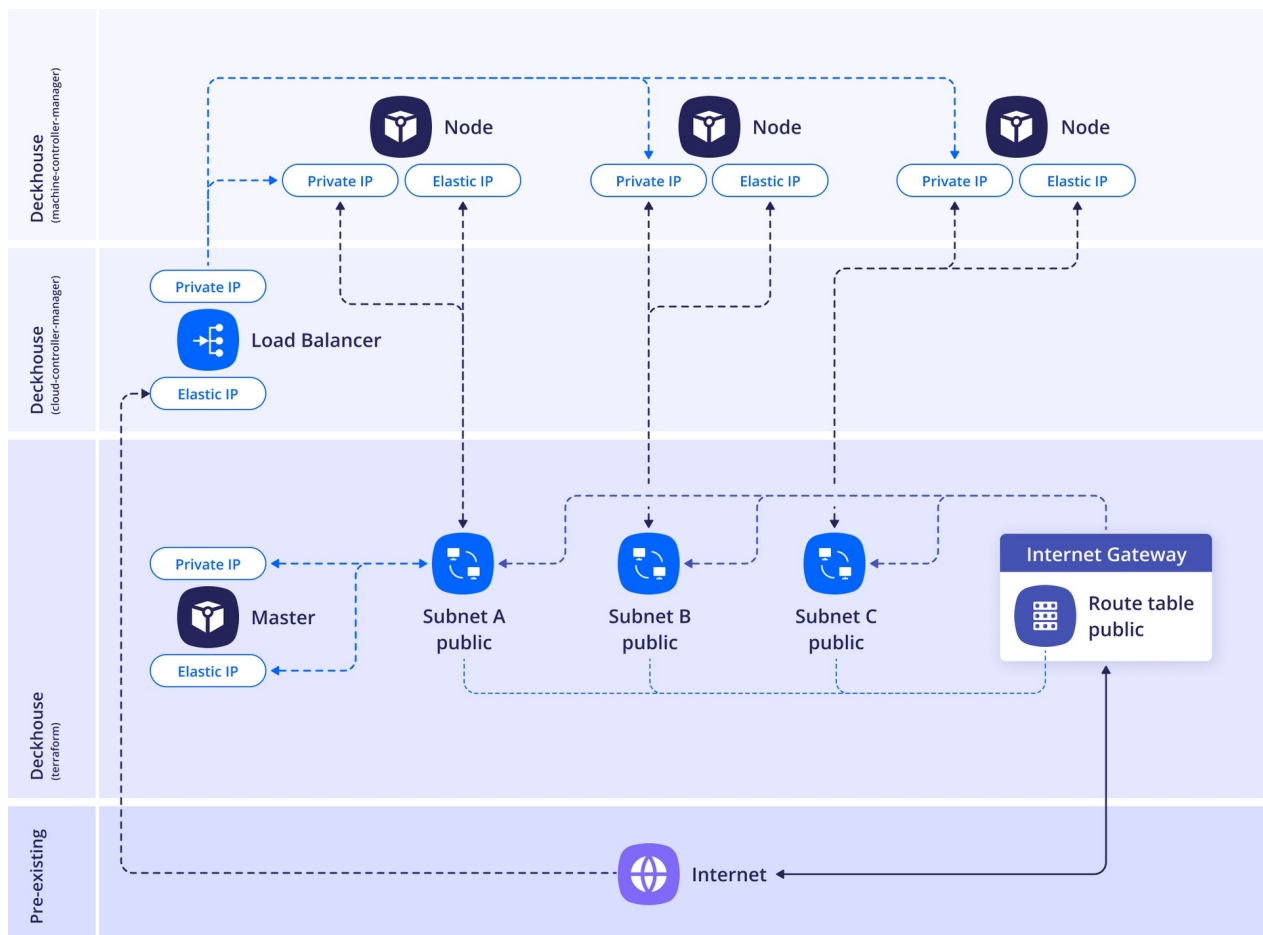
Схемы размещения

Поддерживаются две схемы размещения. Ниже подробнее о каждой из них.

WithoutNAT

Рекомендованная схема размещения.

Каждому узлу присваивается публичный IP (ElasticIP). NAT не используется.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  # Количество master-узлов.
  # Если указано больше одного master-узла, то etcd-кластер соберется автоматически.
  replicas: 1
  instanceClass:
    # Тип используемого инстанса.
    instanceType: m5.xlarge
    # ID образа виртуальной машины в Amazon.
    # Каталог AMI в консоли AWS: EC2 -> AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Размер диска для виртуальной машины master-узла.
    diskSizeGb: 30
    # Используемый тип диска для виртуальной машины master-узла.
    diskType: gp3
nodeGroups:
- name: mydb
  nodeTemplate:
    labels:
      node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: torpedo

```

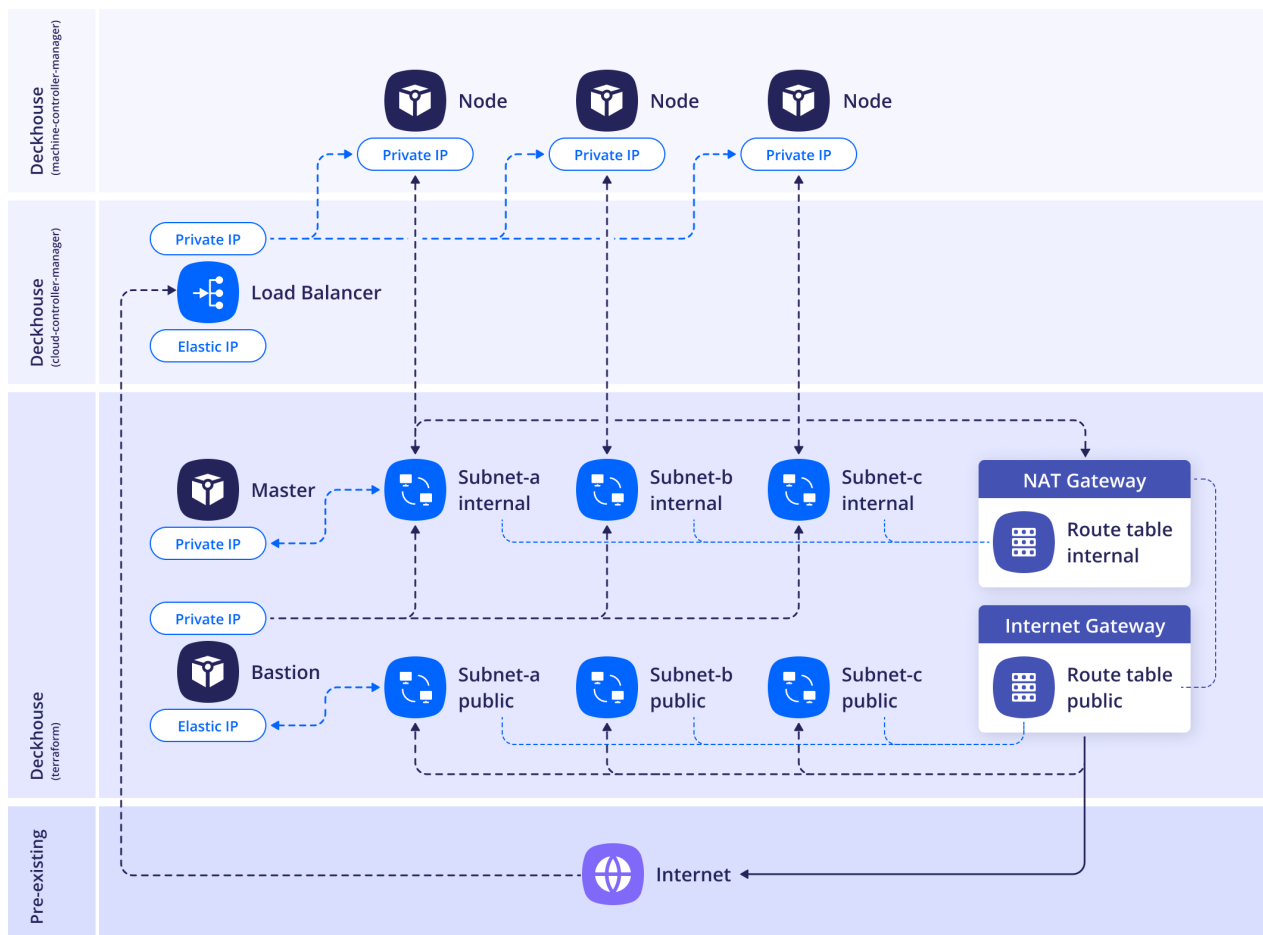
WithNAT

i

Важно! В данной схеме размещения необходим bastion-хост для доступа к узлам (его можно создать вместе с кластером, указав параметры в секции `withNAT.bastionInstance`).

Важно! В этой схеме размещения NAT Gateway всегда создается в зоне `а`. Если узлы кластера будут заказаны в других зонах, при проблемах в зоне `а` они также будут недоступны. Другими словами, при выборе схемы размещения `WithNat` доступность всего кластера будет зависеть от работоспособности зоны `а`.

Виртуальные машины выходят в интернет через NAT Gateway с общим и единственным IP-адресом.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Количество master-узлов.
  # Если указано больше одного master-узла, etcd-кластер соберется автоматически.
  replicas: 1
  instanceClass:
    # Тип используемого инстанса.
    instanceType: m5.xlarge
    # ID образа виртуальной машины в Amazon.
    # Каталог AMI в консоли AWS: EC2 -> AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Размер диска для виртуальной машины master-узла.
    diskSizeGb: 30
    # Используемый тип диска для виртуальной машины master-узла.
    diskType: gp3
nodeGroups:
- name: mydb
  nodeTemplate:
    labels:
      node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  team: torpedo

```

Cloud provider — Azure

Модуль `cloud-provider-azure` обеспечивает интеграцию Deckhouse Kubernetes Platform с Microsoft Azure. Он предоставляет возможность модулю `node-manager` использовать ресурсы Azure при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-azure`:

- Управление ресурсами Azure через `cloud-controller-manager`:
 - создаёт сетевые маршруты для сети PodNetwork на стороне Azure;
 - создаёт балансировщики нагрузки для Service с типом LoadBalancer;
 - актуализирует метаданные узлов кластера и удаляет из Kubernetes узлы, которых больше нет в Azure.
- Заказ дисков через CSI-драйвер Azure Disk (`disk.csi.azure.com`) и создание StorageClass для типов дисков Azure, чтобы из кластера можно было заказывать PersistentVolume.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-azurerm`.
- Заказ CloudEphemeral-узлов через Machine Controller Manager (MCM). Параметры виртуальных машин задаются в ресурсе AzureInstanceClass.
- Регистрация в модуле `node-manager`, чтобы AzureInstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.



Для корректной работы утилит, таких как `ntpd` и `chrony`, убедитесь, что у балансировщика нагрузки есть правила для UDP-трафика. Если исходящий UDP блокируется, добавьте правило к существующему балансировщику или создайте Service с типом LoadBalancer и UDP-портом.

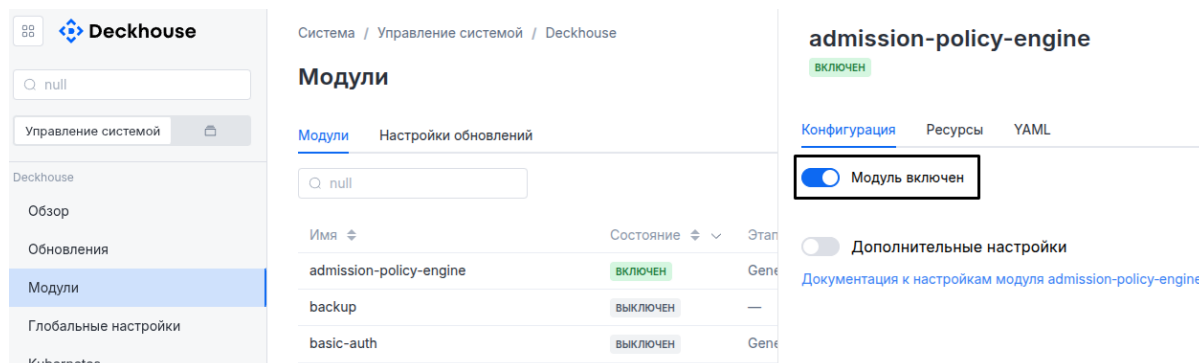
Настройки

Модуль настраивается автоматически, исходя из выбранной схемы размещения (custom resource `AzureClusterConfiguration`). В большинстве случаев нет необходимости ручной конфигурации модуля.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-azure`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-azure`:

```
d8 system module enable cloud-provider-azure
```

- **С помощью ModuleConfig `cloud-provider-azure`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-azure` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-azure`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-azure`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cloud-provider-azure` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-azure`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-azure` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-azure
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-azure`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-azure`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Количество и параметры процесса заказа машин в облаке настраиваются в custom resource `NodeGroup` модуля `node-manager`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.ClassReference`). Инстанс-класс для cloud провайдера Azure — это custom resource `AzureInstanceClass`, в котором указываются конкретные параметры самих машин.

Модуль автоматически создает следующие StorageClass'ы:

Имя	Тип диска
managed-standard-ssd	StandardSSD_LRS
managed-standard	Standard_LRS
managed-premium	Premium_LRS

Также он позволяет сконфигурировать дополнительные StorageClass'ы для дисков с настраиваемыми IOPS и Throughput и отфильтровать ненужные StorageClass'ы, для чего нужно указать их с помощью параметра `exclude`.

Пример конфигурации StorageClass:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
        - name: managed-ultra-ssd
          diskIOPSReadWrite: 600
          diskMBpsReadWrite: 150
      exclude:
        - managed-standard.*
        - managed-premium
      default: managed-ultra-ssd

```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.storageClass**

объект

- **settings.storageClass.default**

Параметр устарел
строка

Имя StorageClass, который будет использоваться в кластере по умолчанию.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- присутствующий StorageClass в кластере, имеющий default-аннотацию;
- первый (по алфавиту) StorageClass из тех, что создаются модулем.

i

Параметр устарел. Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: pd-ssd-not-replicated
```

- **settings.storageClass.exclude**

массив строк

Список полных имен (или регулярных выражений имен) StorageClass, которые не будут созданы в кластере.

Пример:

```

exclude:
- pd-standard.*
- pd-ssd-replicated

```

- **settings.storageClass.provision**

массив объектов

Дополнительные StorageClass'ы для Azure ultra disks.

- **settings.storageClass.provision.cachingMode**

строка

Желаемый режим кеширования. Если размер диска превышает 4 TiB, необходимо использовать `cachineMode: None`.

Дополнительную информацию можно найти в документации Azure.

Допустимые значения: `None`, `ReadOnly`, `ReadWrite`

- **settings.storageClass.provision.diskIOPSReadWrite**

число

Обязательный параметр

Количество IOPS (лимит 300 IOPS/GiB, максимум 160 K IOPS на диск).

- **settings.storageClass.provision.diskMBpsReadWrite**
число

Обязательный параметр

Скорость обращения к диску в MBps (лимит 256 KiB/s на каждый IOPS).

- **settings.storageClass.provision.name**
строка

Обязательный параметр

Имя будущего класса.

- **settings.storageClass.provision.tags**
массив объектов

Azure disk теги.

Пример:

```
tags:
- key: key1
  value: val1
- key: key2
  value: val2
```

- **settings.storageClass.provision.tags.key**
строка

Шаблон: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

- **settings.storageClass.provision.tags.value**
строка

Шаблон: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

- **settings.storageClass.provision.type**
строка

Обязательный параметр

Тип Azure диска.

Дополнительную информацию можно найти в документации Azure.

Допустимые значения: `UltraSSD_LRS`

Настройки провайдера

AzureClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Azure.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```



После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```

apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
vNetCIDR: 10.0.0.0/16
nameservers:
  addresses:
    - 8.8.8.8
    - 8.8.4.4
subnetCIDR: 10.0.0.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202207120
    enableExternalIP: false
provider:
  subscriptionId: "<SUBSCRIPTION_ID>"
  location: westeurope
  clientId: "<CLIENT_ID>"
  clientSecret: "<CLIENT_SECRET>"
  tenantId: "<TENANT_ID>"

```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**

строка

Обязательный параметр

Допустимые значения: `AzureClusterConfiguration`

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

Допустимые значения: `Standard`

- **masterNodeGroup**

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `AzureInstanceClass`.

- **masterNodeGroup.instanceClass.acceleratedNetworking**

булевый

Accelerated Networking обеспечивает пропускную способность сети до 30 Гбит/с.

По умолчанию: `false`

- **masterNodeGroup.instanceClass.additionalTags**

объект

Список дополнительных тегов в формате `key: value`, которые будут назначены инстансам.

Пример:

```
project: cms-production
severity: critical
```

- **masterNodeGroup.instanceClass.diskSizeGb**
целочисленный

Размер root-диска. Значение указывается в Гиб.

Пример:

```
diskSizeGb: 40
```

- **masterNodeGroup.instanceClass.diskType**
строка

Тип создаваемых дисков (root-диска и диска для etcd).

Список доступных типов дисков можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope --zone
```

Пример:

```
diskType: StandardSSD_LRS
```

- **masterNodeGroup.instanceClass.enableExternalIP**
булевый

Разрешение назначения внешнего IP-адреса для инстанса.

Параметр доступен только для layout `Standard`.

По умолчанию: `false`

Допустимые значения: `true`, `false`

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
целочисленный

Размер диска для etcd. Значение указывается в Гиб.

По умолчанию: `20`

Пример:

```
etcdDiskSizeGb: 20
```

- **masterNodeGroup.instanceClass.machineSize**
строка

Обязательный параметр

Тип заказываемых инстансов.

Узнать все доступные типы в регионе можно с помощью Azure CLI:

```
az vm list-sizes --location westeurope -o table
```

Пример:

```
machineSize: Standard_F4
```

- **masterNodeGroup.instanceClass.urn**
строка

Обязательный параметр

Образ виртуальной машины, который будет использоваться для инстанса.

Подробнее про образы виртуальных машин можно прочитать в официальной документации Azure.

Получить `urn` образа с помощью Azure CLI (столбец №4):

```
az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o
table
az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
```

По умолчанию используется образ из `AzureCloudDiscoveryData` — образ, из которого был создан мастер-узел при разворачивании кластера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

- **masterNodeGroup.replicas**
целочисленный

Обязательный параметр

Количество создаваемых master-узлов.

Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **masterNodeGroup.zones**
массив строк

Список зон, в которых допустимо создавать master-узлы.

Доступные зоны для выбранного типа инстанса можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope -o table
```

По умолчанию: `[1,2,3]`

- **nameservers**
объект

Конфигурация DNS-серверов для виртуальной сети Azure, которая используется для узлов кластера.

- **nameservers.addresses**
массив строк

Список DNS-серверов.

Пример:

```
addresses:
- 8.8.8.8
- 8.8.4.4
```

- **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **nodeGroups**
массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**
объект

Обязательный параметр

Частичное содержимое полей AzureInstanceClass.

- **nodeGroups.instanceClass.acceleratedNetworking**
булевый

Accelerated Networking обеспечивает пропускную способность сети до 30 Гбит/с.

По умолчанию: `false`

- **nodeGroups.instanceClass.additionalTags**
объект

Список дополнительных тегов в формате `key: value`, которые будут назначены инстансам.

Пример:

```
project: cms-production
severity: critical
```

- **nodeGroups.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

Пример:

```
diskSizeGb: 40
```

- **nodeGroups.instanceClass.diskType**

строка

Тип создаваемых дисков (root-диска и диска для etcd).

Список доступных типов дисков можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope --zone
```

Пример:

```
diskType: StandardSSD_LRS
```

- **nodeGroups.instanceClass.enableExternalIP**

булевый

Разрешение назначения внешнего IP-адреса для инстанса.

Параметр доступен только для layout `Standard`.

По умолчанию: `false`

Допустимые значения: `true`, `false`

- **nodeGroups.instanceClass.machineSize**

строка

Обязательный параметр

Тип заказываемых инстансов.

Узнать все доступные типы в регионе можно с помощью Azure CLI:

```
az vm list-sizes --location westeurope -o table
```

Пример:

```
machineSize: Standard_F4
```

- **nodeGroups.instanceClass.urn**

строка

Обязательный параметр

Образ виртуальной машины, который будет использоваться для инстанса.

Подробнее про образы виртуальных машин можно прочитать в официальной документации Azure.

Получить `urn` образа с помощью Azure CLI (столбец №4):

```
az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o table
az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
```

По умолчанию используется образ из `AzureCloudDiscoveryData` — образ, из которого был создан мастер-узел при развертывании кластера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: "true"
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если ресурсы это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **nodeGroups.zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Доступные зоны для выбранного типа инстанса можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope -O table
```

По умолчанию: `[1,2,3]`

- **peeredVNETs**

массив объектов

Массив VNet, с которыми будет объединена сеть кластера.

ServiceAccount должен иметь доступ ко всем перечисленным VNet. Если доступа нет, пиринговое соединение необходимо настраивать вручную.

- **peeredVNets.resourceGroupName**
строка

Обязательный параметр

Имя группы ресурсов, в которой находится VNet.

- **peeredVNets.vnetName**
строка

Обязательный параметр

Имя VNet.

- **provider**
объект

Обязательный параметр

Параметры подключения к API Azure.

- **provider.clientId**
строка

Обязательный параметр

Идентификатор клиента.

- **provider.clientSecret**
строка

Обязательный параметр

Секрет (secret) клиента.

Помните о сроке действия секрета. По умолчанию он действителен один год. Чтобы создать service account с большим сроком действия секрета, обратитесь к официальной документации.

- **provider.location**
строка

Обязательный параметр

Имя географического региона, в котором будут созданы все ресурсы. Получение списка доступных локаций через Azure CLI:

```
az account list-locations -O table
```

- **provider.subscriptionId**
строка

Обязательный параметр

Идентификатор подписки.

- **provider.tenantId**
строка

Обязательный параметр

Идентификатор tenant'a.

- **serviceEndpoints**
массив строк

Список Service Endpoints, ассоциированных с подсетью.

Service Endpoints для виртуальной сети позволяют прямо и безопасно подключаться к сервисам Azure по оптимизированному маршруту через магистральную сеть Azure. Они позволяют ограничить пределами отдельных виртуальных сетей доступ к критически важным ресурсам сервисов Azure. Через Service Endpoints приватные IP-адреса в виртуальной сети могут подключаться к эндпоинтам сервисов Azure, не используя публичный IP-адрес в виртуальной сети.

Больше информации можно найти в официальной документации.

- **Элемент массива**
строка

Допустимые значения: `Microsoft.AzureActiveDirectory`, `Microsoft.AzureCosmosDB`, `Microsoft.ContainerRegistry`, `Microsoft.CognitiveServices`, `Microsoft.EventHub`,

Microsoft.KeyVault, Microsoft.ServiceBus, Microsoft.Sql, Microsoft.Storage, Microsoft.Storage.Global, Microsoft.Web

- **sshAllowList**

массив строк

Список CIDR, разрешенных для подключения к узлам по SSH.

По умолчанию `*`.

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы под пользователем `azureuser`.

- **standard**

объект

Настройки для схемы размещения Standard.

- **standard.natGatewayPublicIpCount**

целочисленный

Количество IP-адресов для NAT Gateway (тарификация).

NAT Gateway не используется, если установлено `0`.

По умолчанию: `0`

- **subnetCIDR**

строка

Обязательный параметр

Подсеть из адресного пространства `vNetCIDR`, в которой будут работать узлы кластера.

Виртуальную сеть можно сегментировать на одну или несколько подсетей в пределах ограничений.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|1[1-2][0-9]|0[0-9]))$`

Пример:

```
subnetCIDR: 10.1.2.0/24
```

- **tags**

объект

Словарь тегов, которые будут созданы на всех ресурсах, имеющих такую возможность.

Если поменять теги в рабочем кластере, после применения изменений необходимо пересоздать все машины.

- **vNetCIDR**

строка

Обязательный параметр

Адресное пространство виртуальной сети в формате CIDR.

Виртуальная сеть — это виртуальная изолированная часть общедоступной сети Azure. Каждая виртуальная сеть выделена для вашей подписки.

Внимание. Нужно учитывать, что в случае настройки пиринга, использования VPN или связывания сетей других кластеров, адресное пространство сетей не должно пересекаться.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|1[1-2][0-9]|0[0-9]))$`

Пример:

```
vNetCIDR: 10.0.0.0/16
```

- **zones**

массив строк

Глобальное ограничение набора зон, с которыми работает данный облачный провайдер.

Подготовка окружения



Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.



Внимание! Поддерживаются только регионы, в которых доступны `Availability Zones`.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для управления облаком Microsoft Azure необходимо иметь соответствующую учетную запись и хотя бы одну привязанную подписку (Subscription).

Для управления ресурсами в облаке Microsoft Azure средствами Deckhouse Kubernetes Platform необходимо создать service account. Для этого:

1. Установите Azure CLI, авторизуйтесь и получите `Subscription ID`:

```
export SUBSCRIPTION_ID=$(az login | jq -r '[0].id')
```

2. Создайте service account:

```
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --name "DeckhouseCANDI"
```

Пример вывода команды:

```
{
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",    <-- используется в параметре
  clientId pecypca AzureClusterConfiguration
  "displayName": "DeckhouseCANDI",
  "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", <-- используется в параметре
  clientSecret pecypca AzureClusterConfiguration
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"    <-- используется в параметре
  tenantId pecypca AzureClusterConfiguration
}
```



По умолчанию срок действия секрета созданного service account (используется в параметре `clientSecret` ресурса `AzureClusterConfiguration`) — один год без автоматического продления. Чтобы создать service account с большим сроком действия секрета обратитесь к официальной документации.

Для дальнейшей работы с утилитой `az` необходимо авторизоваться, используя данные (login, password, tenant) созданного service account:

```
az login --service-principal -u <username> -p <password> --tenant <tenant>
```

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

AzureInstanceClass

Scope: Cluster

- v1
- v1alpha1

Параметры группы Azure Instances, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.acceleratedNetworking**
булевый

Accelerated Networking обеспечивает пропускную способность сети до 30 Гбит/с.

По умолчанию: `true`

- **spec.additionalTags**
объект

Дополнительные теги, которые будут присвоены созданным инстансам.

- **spec.capacity**
Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `machineSize`).

Используется `cluster-autoscaler`’ом при планировании, только когда в `NodeGroup`’е еще нет узлов (при `minPerZone` равном 0). Если в `NodeGroup` уже есть узлы, `cluster-autoscaler` использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`Standard_A1_v2`, `Standard_B16ms` и т. п.) заложены в Deckhouse.

- **spec.capacity.cpu**
строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

- **spec.capacity.memory**
строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.diskSizeGb**
целочисленный

Размер root-диска. Значение указывается в ГиБ.

По умолчанию: `50`

Пример:

```
diskSizeGb: 40
```

- **spec.diskType**

строка

Тип созданного диска.

Список доступных типов дисков можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope --zone`
```

По умолчанию: `StandardSSD_LRS`

- **spec.machineSize**

строка

Обязательный параметр

Тип заказываемых инстансов.

Список всех доступных типов в регионе можно посмотреть с помощью Azure CLI:

```
az vm list-sizes --location westeurope -o table
```

Пример:

```
machineSize: Standard_F4
```

- **spec.urn**

строка

Необязательный параметр.

Образ виртуальной машины, который будет использоваться для инстанса.

Подробнее про образы виртуальных машин можно посмотреть в официальной документации.

Последнюю доступную версию образа можно получить с помощью Azure CLI:

- `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope;`
- `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope.`

По умолчанию используется образ из `AzureCloudDiscoveryData` (образ, из которого был создан мастер-узел при разворачивании кластера).

Список поддерживаемых ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

Пример:

```
urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

Параметры группы Azure Instances, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**

объект

Обязательный параметр

- **spec.acceleratedNetworking**

булевый

Accelerated Networking обеспечивает пропускную способность сети до 30 Гбит/с.

По умолчанию: `true`

- **spec.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданным инстансам.

- **spec.capacity**

Параметр устарел

объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `machineSize`).

Используется `cluster-autoscaler`’ом при планировании, только когда в `NodeGroup`’е еще нет узлов (при

minPerZone равно 0). Если в NodeGroup уже есть узлы, cluster-autoscaler использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`Standard_A1_v2`, `Standard_B16ms` и т. п.) заложены в Deckhouse.

- **spec.capacity.cpu**

строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

- **spec.capacity.memory**

строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

По умолчанию: `50`

Пример:

```
diskSizeGb: 40
```

- **spec.diskType**

строка

Тип созданного диска.

Список доступных типов дисков можно посмотреть с помощью Azure CLI:

```
az vm list-skus -l westeurope --zone`
```

По умолчанию: `StandardSSD_LRS`

- **spec.machineSize**

строка

Обязательный параметр

Тип заказываемых инстансов.

Список всех доступных типов в регионе можно посмотреть с помощью Azure CLI:

```
az vm list-sizes --location westeurope -O table
```

Пример:

```
machineSize: Standard_F4
```

- **spec.urn**

строка

Необязательный параметр.

Образ виртуальной машины, который будет использоваться для инстанса.

Подробнее про образы виртуальных машин можно посмотреть в официальной документации.

Последнюю доступную версию образа можно получить с помощью Azure CLI:

- `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope;`
- `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope.`

По умолчанию используется образ из `AzureCloudDiscoveryData` (образ, из которого был создан мастер-узел при развертывании кластера).

Список поддерживаемых ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

Пример:

```
urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
```

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

Примеры

Пример custom resource `AzureInstanceClass`

Ниже представлен простой пример custom resource `AzureInstanceClass`:

```
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: example
spec:
  machineSize: Standard_F4
```

Схемы размещения

Внимание. Поддерживаются только те регионы, в которых доступны `Availability Zones`.

Standard

- Для кластера создается отдельная ресурсная группа (resource group).
- По умолчанию каждому экземпляру динамически выделяется один внешний IP-адрес, который используется только для доступа в интернет. На каждый IP-адрес для SNAT доступно 64000 портов.
- Поддерживается NAT Gateway (тарификация). Она позволяет использовать статические публичные IP-адреса для SNAT.
- Публичные IP-адреса можно назначить как на master-узлы, так и на узлы, созданные Terraform.
- Если master-узел не имеет публичного IP-адреса, для установки и доступа в кластер необходим дополнительный инстанс с публичным IP-адресом (например, bastion-хост). В этом случае также потребуется настроить пиринговое соединение между VNet кластера и VNet bastion-хоста.
- Между VNet кластера и другими VNet можно настроить пиринговое соединение.

Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>" # Обязательный параметр.
vNetCIDR: 10.50.0.0/16 # Обязательный параметр.
subnetCIDR: 10.50.0.0/24 # Обязательный параметр.
standard:
  natGatewayPublicIpCount: 1 # Необязательный параметр, по умолчанию 0.
masterNodeGroup:
  replicas: 1
  zones: ["1"] # Необязательный параметр, по умолчанию ["1"].
instanceClass:
  machineSize: Standard_F4 # Обязательный параметр.
  diskSizeGb: 32
  urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140 # Обязательный параметр.
  enableExternalIP: false # Необязательный параметр, по умолчанию true.
provider:
  subscriptionId: "<SUBSCRIPTION_ID>" # Обязательный параметр.
  clientId: "<CLIENT_ID>" # Обязательный параметр.
  clientSecret: "<CLIENT_SECRET>" # Обязательный параметр.
  tenantId: "<TENANT_ID>" # Обязательный параметр.
  location: "westeurope" # Обязательный параметр.
# Необязательный параметр, список Azure VNets, с которыми Kubernetes VNet будет
# соединяться через пиринговое соединение.
peeredVNets:
  - resourceGroupName: kube-bastion # Обязательный параметр.
    vnetName: kube-bastion-vnet # Обязательный параметр.
```

Cloud provider — DVP

Модуль `cloud-provider-dvp` обеспечивает интеграцию Deckhouse Kubernetes Platform с Deckhouse Virtualization Platform. Он предоставляет возможность модулю `node-manager` использовать ресурсы DVP при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-dvp`:

- Управление ресурсами DVP через `cloud-controller-manager`: актуализирует метаданные виртуальных машин и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в DVP.
- Заказ дисков через CSI-драйвер DVP (`csi.dvp.deckhouse.io`), чтобы из кластера можно было заказывать PersistentVolume.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-kubernetes`.
- Заказ CloudEphemeral-узлов через Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе DVPIstanceClass.
- Регистрация в модуле `node-manager`, чтобы DVPIstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.



Если кластер был установлен со схемой DVPClusterConfiguration, необходима миграция на конфигурацию через ModuleConfig. Пока миграция не выполнена, может срабатывать алерт D8CloudProviderDVPMigrationPending, а обновление Deckhouse — блокироваться.

Инструкция: Как мигрировать облачный провайдер на конфигурацию через ModuleConfig.

Настройки

Модуль автоматически включается для всех облачных кластеров, развернутых в DVP.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dvp`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left, the 'Modules' section is selected in the sidebar. The main area displays a list of modules with columns for 'Имя' (Name), 'Состояние' (Status), and 'Этап' (Stage). The 'admission-policy-engine' module is highlighted, and its status is 'ВКЛЮЧЕН' (Enabled). The 'cloud-provider-dvp' module is also visible in the list. On the right, the 'admission-policy-engine' configuration page is shown, with the 'Модуль включен' (Module enabled) toggle switch checked.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-dvp`:

```
d8 system module enable cloud-provider-dvp
```

- **С помощью ModuleConfig `cloud-provider-dvp`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-dvp` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-dvp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dvp`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы

на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `cloud-provider-dvp` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dvp`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-dvp` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-dvp
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-dvp`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-dvp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.provider**
объект

Содержит параметры подключения к API Deckhouse Kubernetes Platform.

- **settings.provider.kubeconfigDataBase64**
строка

Обязательный параметр

Kubeconfig для подключения к API Deckhouse Kubernetes Platform, закодированный в base64.

Пример:

```
kubeconfigDataBase64: ZXhhbXBsZQo=
```

- **settings.provider.namespace**

строка

Обязательный параметр

Namespace, в котором будут создаваться ресурсы кластера DKP.

i Внимание! Если значение явно не указано, будет использован namespace по умолчанию из kubeconfig.

Пример:

```
namespace: default
```

- **settings.zones**

массив строк

Набор зон, в которых могут создаваться узлы. Для использования этой настройки на узлах DVP должна быть установлена метка `topology.kubernetes.io/zone`. Подробнее о топологических метках см. в документации Kubernetes.

i Внимание! Чтобы задать необходимую метку для узла DVP, следуйте документации по NodeGroup.

Пример:

```
zones:
- zone-a
- zone-b
```

Настройки провайдера

DVPClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Deckhouse Virtualization Platform (DVP).

Используется облачным провайдером, если управляющий слой (control plane) кластера размещён в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i Чтобы изменения вступили в силу, после изменения параметров узлов выполните команду `dhctl converge`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- zone-a
- zone-b
- zone-c
region: r1
masterNodeGroup:
  replicas: 3
  zones:
  - zone-a
  - zone-b
  - zone-c
instanceClass:
  virtualMachine:
    cpu:
      cores: 1
      coreFraction: 100%
    memory:
      size: 4Gi
  virtualMachineClassName: generic
ipAddresses:
- 10.66.30.100
- 10.66.30.101
```

```

- 10.66.30.102
additionalLabels:
  additional-vm-label: label-value
additionalAnnotations:
  additional-vm-annotation: annotation-value
tolerations:
- key: dedicated.deckhouse.io
  operator: Equal
  value: system
nodeSelector:
  beta.kubernetes.io/os: linux
rootDisk:
  size: 10Gi
  storageClass: linstor-thin-r1
image:
  kind: ClusterVirtualImage
  name: ubuntu-2204
etcdDisk:
  size: 10Gi
  storageClass: linstor-thin-r1
additionalDisks:
- size: 10Gi
  storageClass: linstor-thin-r1
- size: 1Gi
  storageClass: linstor-thin-r1
nodeGroups:
- name: worker
  zones:
  - zone-a
  - zone-b
  - zone-c
replicas: 1
instanceClass:
  virtualMachine:
    cpu:
      cores: 4
      coreFraction: 100%
    memory:
      size: 8Gi
    virtualMachineClassName: generic
rootDisk:
  size: 10Gi
  image:
    kind: ClusterVirtualImage
    name: ubuntu-2204
additionalDisks:
- size: 10Gi
  storageClass: linstor-thin-r1
- size: 1Gi
  storageClass: linstor-thin-r1
provider:
  kubeconfigDataBase64: ZXhhbXBsZQo=
  namespace: default

```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`

- **kind**

строка

Обязательный параметр

Допустимые значения: `DVPClusterConfiguration`

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

Допустимые значения: `Standard`

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Параметры виртуальной машины и ее дисков для созданного master-узла.

- **masterNodeGroup.instanceClass.additionalDisks**

массив объектов

Параметры дополнительных дисков виртуальной машины.

Каждый элемент массива описывает отдельный дополнительный диск. Для каждого диска необходимо задать параметры `size` и `storageClass`.

- **masterNodeGroup.instanceClass.additionalDisks.size**

строка

Обязательный параметр

Размер дополнительного диска.

Шаблон: `^(\+|-)?([0-9]+(\.[0-9]*)?)|(\.[0-9]+))((([KMGTPE]i)|[numkMGTPE])|([eE](\+|-)?([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

Пример:

```
size: 10Gi
```

- **masterNodeGroup.instanceClass.additionalDisks.storageClass**

строка

Обязательный параметр

Имя существующего StorageClass, который будет использоваться для создания дополнительного диска.

Пример:

```
storageClass: ceph-pool-r2-csi-rbd-immediate
```

- **masterNodeGroup.instanceClass.additionalLabels**

объект

Дополнительные лейблы.

Пример:

```
project: cms-production
severity: critical
```

- **masterNodeGroup.instanceClass.etcdDisk**

объект

Обязательный параметр

Указание настроек для диска данных etcd.

- **masterNodeGroup.instanceClass.etcdDisk.size**

строка

Обязательный параметр

Размер диска etcd.

Шаблон: `^(\+|-)?([0-9]+(\.[0-9]*)?)|(\.[0-9]+))((([KMGTPE]i)|[numkMGTPE])|([eE](\+|-)?([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

Пример:

```
size: 10Gi
```

- **masterNodeGroup.instanceClass.etcdDisk.storageClass**

строка

Имя существующего StorageClass будет использоваться для создания диска данных etcd.

Если значение не указано, то будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

■ **masterNodeGroup.instanceClass.rootDisk** объект

Обязательный параметр

Указывает настройки для корневого диска виртуальной машины.

■ **masterNodeGroup.instanceClass.rootDisk.image** объект

Обязательный параметр

Параметры образа, который будет использоваться для создания корневого диска виртуальной машины.

■ **masterNodeGroup.instanceClass.rootDisk.image.kind** строка

Обязательный параметр

Тип источника изображения.

Допустимые значения: `ClusterVirtualImage`, `VirtualImage`

■ **masterNodeGroup.instanceClass.rootDisk.image.name** строка

Обязательный параметр

Имя образа, который будет использоваться для создания корневого диска.

i Для установки требуются образы ОС Linux с предустановленным cloud-init.

■ **masterNodeGroup.instanceClass.rootDisk.size** строка

Обязательный параметр

Размер корневого диска.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))([KMGTPeI]| [numkMGTPeI]| ([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))))?$`

Пример:

```
size: 10Gi
```

■ **masterNodeGroup.instanceClass.rootDisk.storageClass** строка

Имя существующего StorageClass будет использоваться для создания корневого диска виртуальной машины.

Если значение не указано, то будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

■ **masterNodeGroup.instanceClass.virtualMachine** объект

Обязательный параметр

Настройки виртуальной машины для созданного master-узла.

i Для виртуальных машин узлов используется политика запуска `runPolicy: AlwaysOnUnlessStoppedManually`. Это позволяет вручную останавливать ВМ (например, для обслуживания) без принудительного автозапуска.

■ **masterNodeGroup.instanceClass.virtualMachine.additionalAnnotations** объект

Дополнительные аннотации для ресурса виртуальной машины.

Пример:

```
cluster-owner: user
```

■ **masterNodeGroup.instanceClass.virtualMachine.additionalLabels** объект

Дополнительные лейблы для ресурса виртуальной машины.

Пример:

```
cluster-owner: user
```

■ **masterNodeGroup.instanceClass.virtualMachine.bootloader**
строка

Определяет загрузчик для виртуальной машины.

- BIOS — используется BIOS;
- EFI — используется Unified Extensible Firmware (EFI/UEFI);
- EFIWithSecureBoot — используется UEFI/EFI с поддержкой Secure Boot.

По умолчанию: `EFI`

Допустимые значения: `BIOS`, `EFI`, `EFIWithSecureBoot`

■ **masterNodeGroup.instanceClass.virtualMachine.cpu**
объект

Обязательный параметр

Настройки процессора для виртуальной машины.

■ **masterNodeGroup.instanceClass.virtualMachine.cpu.coreFraction**
строка

Процент гарантированной доли CPU, которая будет выделена для виртуальной машины.

По умолчанию: `100%`

Шаблон: `^100%$|^[1-9][0-9]?%$`

Пример:

```
coreFraction: 100%
```

■ **masterNodeGroup.instanceClass.virtualMachine.cpu.cores**
целочисленный

Обязательный параметр

Количество ядер процессора для виртуальной машины.

Допустимые значения: `1 <= X`

■ **masterNodeGroup.instanceClass.virtualMachine.ipAddresses**
массив строк

Статические IP-адреса, назначаемые сетевым интерфейсам виртуальных машин. Количество адресов должно соответствовать числу создаваемых реплик — каждый IP-адрес будет закреплён за конкретной репликой. Например, если задано 3 реплики и указаны IP-адреса: ip1, ip2, ip3, то ip1 будет назначен первой реплике, ip2 — второй, а ip3 — третьей.



Эти адреса должны принадлежать диапазону адресов, указанному в конфигурации модуля виртуализации в параметре `virtualMachineCIDRs`.

■ **Элемент массива**
строка

Шаблон: `^([0-9]{1,3}\.){0-3}[0-9]{1,3}([0-9]{1,3})?(Auto)$`

■ **masterNodeGroup.instanceClass.virtualMachine.liveMigrationPolicy**
строка

Политика живой миграции виртуальной машины.

- Manual — миграция управляется вручную;
- Never — миграция отключена;
- AlwaysSafe — всегда использовать безопасную миграцию (может не сработать при высокой скорости изменений памяти VM);
- PreferSafe — предпочитать безопасную миграцию, переключаться на forced при необходимости;
- AlwaysForced — всегда использовать forced-миграцию с замедлением VM;
- PreferForced — предпочитать forced-миграцию (рекомендуется для master-узлов из-за высокой активности памяти).

По умолчанию: `PreferForced`

Допустимые значения: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

■ **masterNodeGroup.instanceClass.virtualMachine.memory**
объект

Обязательный параметр

Определяет параметры памяти для виртуальной машины.

■ **masterNodeGroup.instanceClass.virtualMachine.memory.size**
строка

Обязательный параметр

Количество ресурсов памяти, разрешенных для виртуальной машины.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))([KMGTPE]i)|[numkMGTPe]|([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+)))?&`

Пример:

```
size: 4Gi
```

■ **masterNodeGroup.instanceClass.virtualMachine.nodeSelector**

объект

Позволяет назначить виртуальную машину на указанные узлы DVP. Подробнее — как в параметре `spec.nodeSelector` подов в Kubernetes.

■ **masterNodeGroup.instanceClass.virtualMachine.priorityClassName**

строка

Подробнее — как в параметре `spec.priorityClassName` подов в Kubernetes.

■ **masterNodeGroup.instanceClass.virtualMachine.runPolicy**

строка

Политика запуска виртуальной машины.

- `AlwaysOn` — виртуальная машина всегда должна быть запущена;
- `AlwaysOff` — виртуальная машина всегда должна быть остановлена;
- `Manual` — состояние виртуальной машины управляется вручную;
- `AlwaysOnUnlessStoppedManually` — виртуальную машину можно остановить вручную (например, для обслуживания), но она автоматически запустится после перезагрузки хоста.

По умолчанию: `AlwaysOnUnlessStoppedManually`

Допустимые значения: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

■ **masterNodeGroup.instanceClass.virtualMachine.tolerations**

массив объектов

Позволяет установить допуски для виртуальной машины для узла DVP. Подробнее — как в параметре `spec.tolerations` подов в Kubernetes;

- **masterNodeGroup.instanceClass.virtualMachine.tolerations.effect**
строка
- **masterNodeGroup.instanceClass.virtualMachine.tolerations.key**
строка
- **masterNodeGroup.instanceClass.virtualMachine.tolerations.operator**
строка
- **masterNodeGroup.instanceClass.virtualMachine.tolerations.tolerationSeconds**
целочисленный
- **masterNodeGroup.instanceClass.virtualMachine.tolerations.value**
строка

■ **masterNodeGroup.instanceClass.virtualMachine.virtualMachineClassName**

строка

Обязательный параметр

Имя `VirtualMachineClass`.

Предназначен для централизованной конфигурации предпочтительных параметров виртуальных машин. Ресурс позволяет задавать инструкции CPU, политики конфигурации ресурсов CPU и памяти для виртуальных машин, а также устанавливать соотношения этих ресурсов.

○ **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов.

Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

○ **masterNodeGroup.zones**

массив строк

Набор зон, в которых могут быть созданы узлы.

Чтобы использовать эту настройку, на узлах DVP должна быть установлен лейбл `topology.kubernetes.io/zone`. Подробнее о топологических лейблах.

i

Чтобы установить требуемый лейбл для узла DVP, обратитесь к документации по NodeGroup.

• **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Параметры виртуальной машины и диска для созданного статического узла.

- **nodeGroups.instanceClass.additionalDisks**

массив объектов

Параметры дополнительных дисков виртуальной машины.

Каждый элемент массива описывает отдельный дополнительный диск. Для каждого диска необходимо задать параметры `size` и `storageClass`.

- **nodeGroups.instanceClass.additionalDisks.size**

строка

Обязательный параметр

Размер дополнительного диска.

Шаблон: `^(\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+))((([KMGTPE]i)|[numkMGTP])|([eE](\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))))?)?$`

Пример:

```
size: 10Gi
```

- **nodeGroups.instanceClass.additionalDisks.storageClass**

строка

Обязательный параметр

Имя существующего StorageClass, который будет использоваться для создания дополнительного диска.

Пример:

```
storageClass: ceph-pool-r2-csi-rbd-immediate
```

- **nodeGroups.instanceClass.rootDisk**

объект

Обязательный параметр

Указывает настройки для корневого диска виртуальной машины.

- **nodeGroups.instanceClass.rootDisk.image**

объект

Обязательный параметр

Параметры образа, который будет использоваться для создания корневого диска виртуальной машины.

- **nodeGroups.instanceClass.rootDisk.image.kind**

строка

Обязательный параметр

Тип источника изображения.

Допустимые значения: `ClusterVirtualImage`, `VirtualImage`

- **nodeGroups.instanceClass.rootDisk.image.name**

строка

Обязательный параметр

Имя образа, который будет использоваться для создания корневого диска.

i

Для установки требуются образы ОС Linux с предустановленным cloud-init.

- **nodeGroups.instanceClass.rootDisk.size**

строка

Обязательный параметр

Размер корневого диска.

Шаблон: `^(\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+))((([KMGTPE]i)|[numkMGTP])|([eE](\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))))?)?$`

Пример:

```
size: 10Gi
```

■ **nodeGroups.instanceClass.rootDisk.storageClass**

строка

Имя существующего StorageClass будет использоваться для создания корневого диска виртуальной машины.

Если значение не указано, то будет использоваться StorageClass, согласно настройке глобального параметра storageClass.

■ **nodeGroups.instanceClass.virtualMachine**

объект

Обязательный параметр

Настройки виртуальной машины для созданного master-узла.

i

Для виртуальных машин узлов используется политика запуска `runPolicy: AlwaysOnUnlessStoppedManually`. Это позволяет вручную останавливать ВМ (например, для обслуживания) без принудительного автозапуска.

■ **nodeGroups.instanceClass.virtualMachine.additionalAnnotations**

объект

Дополнительные аннотации для ресурса виртуальной машины.

Пример:

```
cluster-owner: user
```

■ **nodeGroups.instanceClass.virtualMachine.additionalLabels**

объект

Дополнительные лейблы для ресурса виртуальной машины.

Пример:

```
cluster-owner: user
```

■ **nodeGroups.instanceClass.virtualMachine.bootloader**

строка

Определяет загрузчик для виртуальной машины.

- BIOS — используется BIOS;
- EFI — используется Unified Extensible Firmware (EFI/UEFI);
- EFIWithSecureBoot — используется UEFI/EFI с поддержкой Secure Boot.

По умолчанию: `EFI`

Допустимые значения: `BIOS`, `EFI`, `EFIWithSecureBoot`

■ **nodeGroups.instanceClass.virtualMachine.cpu**

объект

Обязательный параметр

Настройки процессора для виртуальной машины.

■ **nodeGroups.instanceClass.virtualMachine.cpu.coreFraction**

строка

Процент гарантированной доли CPU, которая будет выделена для виртуальной машины.

По умолчанию: `100%`

Шаблон: `^100%$|^[1-9][0-9]?%$`

Пример:

```
coreFraction: 100%
```

■ **nodeGroups.instanceClass.virtualMachine.cpu.cores**

целочисленный

Обязательный параметр

Количество ядер процессора для виртуальной машины.

Допустимые значения: `1 <= X`

■ **nodeGroups.instanceClass.virtualMachine.ipAddresses**

массив строк

Статические IP-адреса, назначаемые сетевым интерфейсам виртуальных машин. Количество адресов должно соответствовать числу создаваемых реплик — каждый IP-адрес будет закреплён за конкретной репликой. Например, если задано 3 реплики и указаны IP-адреса: ip1, ip2, ip3, то ip1 будет назначен первой реплике, ip2 — второй, а ip3 — третьей.

i Эти адреса должны принадлежать диапазону адресов, указанному в конфигурации модуля виртуализации в параметре `virtualMachineCIDRs`.

- **Элемент массива**
строка

Шаблон: `^([0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3})|(Auto)$`

- **nodeGroups.instanceClass.virtualMachine.liveMigrationPolicy**
строка

Политика живой миграции виртуальной машины.

- `Manual` — миграция управляется вручную;
- `Never` — миграция отключена;
- `AlwaysSafe` — всегда использовать безопасную миграцию (может не сработать при высокой скорости изменений памяти VM);
- `PreferSafe` — предпочитать безопасную миграцию, переключаться на `forced` при необходимости;
- `AlwaysForced` — всегда использовать `forced`-миграцию с замедлением VM;
- `PreferForced` — предпочитать `forced`-миграцию (рекомендуется для master-узлов из-за высокой активности памяти).

По умолчанию: `PreferForced`

Допустимые значения: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

- **nodeGroups.instanceClass.virtualMachine.memory**
объект

Обязательный параметр

Определяет параметры памяти для виртуальной машины.

- **nodeGroups.instanceClass.virtualMachine.memory.size**
строка

Обязательный параметр

Количество ресурсов памяти, разрешенных для виртуальной машины.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))([KMGTPE]i|[numkMGTP])|([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+)))?$`

Пример:

```
size: 4Gi
```

- **nodeGroups.instanceClass.virtualMachine.nodeSelector**
объект

Позволяет назначить виртуальную машину на указанные узлы DVP. Подробнее — как в параметре `spec.nodeSelector` подов в Kubernetes.

- **nodeGroups.instanceClass.virtualMachine.priorityClassName**
строка

Подробнее — как в параметре `spec.priorityClassName` подов в Kubernetes.

- **nodeGroups.instanceClass.virtualMachine.runPolicy**
строка

Политика запуска виртуальной машины.

- `AlwaysOn` — виртуальная машина всегда должна быть запущена;
- `AlwaysOff` — виртуальная машина всегда должна быть остановлена;
- `Manual` — состояние виртуальной машины управляется вручную;
- `AlwaysOnUnlessStoppedManually` — виртуальную машину можно остановить вручную (например, для обслуживания), но она автоматически запустится после перезагрузки хоста.

По умолчанию: `AlwaysOnUnlessStoppedManually`

Допустимые значения: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

- **nodeGroups.instanceClass.virtualMachine.tolerations**
массив объектов

Позволяет установить допуски для виртуальной машины для узла DVP. Подробнее — как в параметре `spec.tolerations` подов в Kubernetes;

- **nodeGroups.instanceClass.virtualMachine.tolerations.effect**
строка

- **nodeGroups.instanceClass.virtualMachine.tolerations.key**
строка
- **nodeGroups.instanceClass.virtualMachine.tolerations.operator**
строка
- **nodeGroups.instanceClass.virtualMachine.tolerations.tolerationSeconds**
целочисленный
- **nodeGroups.instanceClass.virtualMachine.tolerations.value**
строка

- **nodeGroups.instanceClass.virtualMachine.virtualMachineClassName**
строка

Обязательный параметр

Имя VirtualMachineClass.

Предназначен для централизованной конфигурации предпочтительных параметров виртуальных машин. Ресурс позволяет задавать инструкции CPU, политики конфигурации ресурсов CPU и памяти для виртуальных машин, а также устанавливать соотношения этих ресурсов.

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: "true"
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если ресурсы это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если вы меняете лейблы в рабочем кластере, после применения изменений пересоздайте все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **nodeGroups.zones**
массив строк

Набор зон, в которых могут быть созданы узлы.

Чтобы использовать эту настройку, на узлах DVP должна быть установлен лейбл `topology.kubernetes.io/zone`. Подробнее о топологических лейблах.

i Чтобы установить требуемый лейбл для узла DVP, обратитесь к документации по NodeGroup.

- **provider**
объект

Обязательный параметр

Содержит настройки для подключения к API Deckhouse Kubernetes Platform.

- **provider.kubeconfigDataBase64**
строка

Обязательный параметр

Kubeconfig для подключения к Deckhouse Kubernetes Platform API в кодировке Base64.

Шаблон: `^[A-Za-z0-9+/]{0,2}$`

- **provider.namespace**
строка

Обязательный параметр

Пространство имён, в котором будут созданы ресурсы кластера DKP.

i Если не указано явно, будет использоваться пространство имён по умолчанию для kubeconfig.

- **provider.networkPolicy**
строка

Правила управления сетевым трафиком к нагрузкам и от нагрузок, запущенных в рамках ресурса Project.

- **Isolated** — применяется ограничивающая allowlist-политика: разрешён только сетевой трафик, необходимый для работы платформенных компонентов; весь остальной трафик запрещён.
- **None** — кластер не заказывает сетевые политики. Ограничения, заданные на уровне Project, продолжают действовать.

По умолчанию: **Isolated**

Допустимые значения: **Isolated**, **None**

- **region**
строка

Название региона.

Чтобы использовать эту настройку, на узлах DVP должна быть установлен лейбл `topology.kubernetes.io/region`. Подробнее о топологических лейблах

i Чтобы установить требуемый лейбл для узла DVP, следуйте документации по NodeGroup.

- **sshPublicKey**
строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **zones**
массив строк

Набор зон, в которых могут быть созданы узлы.

Чтобы использовать эту настройку, на узлах DVP должна быть установлен лейбл `topology.kubernetes.io/zone`. Подробнее о топологических лейблах.

i Чтобы установить требуемый лейбл для узла DVP, обратитесь к документации по NodeGroup.

Подготовка окружения

Для взаимодействия с ресурсами в DVP компоненты Deckhouse Kubernetes Platform используют API DVP. Для настройки подключения создайте пользователя (ServiceAccount), назначьте ему соответствующие права доступа и

сгенерируйте kubeconfig.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.



Если модуль `update-hostname` пакета `cloud-init` не отключён, рекомендуется изменить частоту его запуска с `always` на `once-per-instance`.

Для этого измените конфигурацию модуля `update-hostname` в файле `/etc/cloud/cloud.cfg`:

```
cloud_init_modules:
...
- [update-hostname, once-per-instance]
...
```

Также модуль `update-hostname` можно полностью отключить, удалив его из списка модулей `cloud_init_modules` в файле `/etc/cloud/cloud.cfg`.

Создание пользователя

Создайте нового пользователя в кластере DVP с помощью следующей команды:

```
d8 k create -f -<<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-demo
  namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: sa-demo-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: sa-demo
type: kubernetes.io/service-account-token
EOF
```

Добавление роли

Добавьте роль созданному пользователю в кластере DVP с помощью следующей команды:

```
d8 k create -f -<<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-demo-rb
  namespace: default
subjects:
- kind: ServiceAccount
  name: sa-demo
  namespace: default
roleRef:
  kind: ClusterRole
  name: d8:use:role:manager
apiGroup: rbac.authorization.k8s.io
EOF
```

Генерация kubeconfig

Сгенерируйте kubeconfig, который будет использоваться в файле первичной конфигурации кластера:

```
cat <<EOF > kubeconfig
apiVersion: v1
clusters:
- cluster:
    server: https://<KUBE-APISERVER-URL> # Замените на реальный адрес API-сервера кластера.
    name: <CLUSTER-NAME> # Замените на имя кластера.
contexts:
- context:
    cluster: <CLUSTER-NAME> # Замените на имя кластера.
    user: sa-demo
    namespace: default
    name: sa-demo-context
current-context: sa-demo-context
kind: Config
preferences: {}
users:
- name: sa-demo
  user:
    token: $(d8 k get secret sa-demo-token -n default -o json | jq -rC .data.token | base64 -d)
EOF
```

Закодируйте сгенерированный kubeconfig в кодировке Base64 (он указывается в файле первичной конфигурации в таком виде):

```
base64 kubeconfig | tr -d '\n'
```

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

DVPInstanceClass

Scope: Cluster

Version: v1alpha1

- **spec**

объект

Обязательный параметр

- **spec.additionalDisks**

массив объектов

Параметры дополнительных дисков виртуальной машины.

Каждый элемент массива описывает отдельный дополнительный диск. Для каждого диска необходимо задать параметры `size` и `storageClass`.

- **spec.additionalDisks.size**

строка

Обязательный параметр

Размер дополнительного диска.

Шаблон: `^(\+|-)?(((0-9)+(\. (0-9)*)?)|(\. (0-9)+))((([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(((0-9)+(\. (0-9)*)?)|(\. (0-9)+)))))?$`

Пример:

```
size: 10Gi
```

- **spec.additionalDisks.storageClass**

строка

Обязательный параметр

Имя существующего StorageClass, который будет использоваться для создания дополнительного диска.

- **spec.rootDisk**

объект

Обязательный параметр

Указывает настройки для корневого диска виртуальной машины.

- **spec.rootDisk.image**

объект

Обязательный параметр

Параметры образа, который будет использоваться для создания корневого диска виртуальной машины.

- **spec.rootDisk.image.kind**

строка

Обязательный параметр

Тип источника изображения.

Допустимые значения: `ClusterVirtualImage`, `VirtualImage`, `VirtualDisk`

- **spec.rootDisk.image.name**

строка

Обязательный параметр

Имя образа, который будет использоваться для создания корневого диска.

i

Для установки требуются образы ОС Linux с предустановленным cloud-init.

- **spec.rootDisk.size**

строка

Обязательный параметр

Размер корневого диска.

Шаблон: `^(\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+))((([KMGTPE]i)|[numkMGTPE])|([eE](\+|-)?((([0-9]+(\.[0-9]*)?)|(\.[0-9]+)))))?$`

Пример:

```
size: 10Gi
```

- **spec.rootDisk.storageClass**

строка

Имя существующего StorageClass будет использоваться для создания корневого диска виртуальной машины.

Если значение не указано, то будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

- **spec.virtualMachine**

объект

Обязательный параметр

Настройки виртуальной машины для созданного узла.

i

Для виртуальных машин узлов используется политика запуска `runPolicy: AlwaysOnUnlessStoppedManually`. Это позволяет вручную останавливать VM (например, для обслуживания) без принудительного автозапуска.

- **spec.virtualMachine.bootloader**

строка

Определяет загрузчик для виртуальной машины.

- `BIOS` — используется BIOS;
- `EFI` — используется Unified Extensible Firmware (EFI/UEFI);
- `EFIWithSecureBoot` — используется UEFI/EFI с поддержкой Secure Boot.

По умолчанию: `EFI`

Допустимые значения: `BIOS`, `EFI`, `EFIWithSecureBoot`

- **spec.virtualMachine.cpu**

объект

Обязательный параметр

Настройки процессора для виртуальной машины.

- **spec.virtualMachine.cpu.coreFraction**

строка

Процент гарантированной доли CPU, которая будет выделена виртуальной машине.

По умолчанию: `100%`

Шаблон: `^100%|^[1-9][0-9]?%$`

Пример:

```
coreFraction: 100%
```

■ **spec.virtualMachine.cpu.cores**

целочисленный

Обязательный параметр

Количество ядер процессора для виртуальной машины.

Допустимые значения: `1 <= X`

Пример:

```
cores: 4
```

■ **spec.virtualMachine.liveMigrationPolicy**

строка

Политика живой миграции виртуальной машины.

- `Manual` — миграция управляется вручную;
- `Never` — миграция отключена;
- `AlwaysSafe` — всегда использовать безопасную миграцию (может не сработать при высокой скорости изменений памяти ВМ);
- `PreferSafe` — предпочитать безопасную миграцию, переключаться на `forced` при необходимости;
- `AlwaysForced` — всегда использовать `forced`-миграцию с замедлением ВМ;
- `PreferForced` — предпочитать `forced`-миграцию (рекомендуется для master-узлов из-за высокой активности памяти).

По умолчанию: `PreferForced`Допустимые значения: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

Пример:

```
liveMigrationPolicy: PreferForced
```

■ **spec.virtualMachine.memory**

объект

Обязательный параметр

Определяет параметры памяти для виртуальной машины.

■ **spec.virtualMachine.memory.size**

строка

Обязательный параметр

Количество ресурсов памяти, разрешенных для виртуальной машины.

Шаблон: `^(\\+|-)?(((0-9)+(\\. [0-9]*)?)|(\\. [0-9]+))((([KMGTPE]i)|[numkMGTPe])|([eE](\\+|-)?((0-9)+(\\. [0-9]*)?)|(\\. [0-9]+))))?$`

Пример:

```
size: 4Gi
```

■ **spec.virtualMachine.runPolicy**

строка

Политика запуска виртуальной машины.

- `AlwaysOn` — виртуальная машина всегда должна быть запущена;
- `AlwaysOff` — виртуальная машина всегда должна быть остановлена;
- `Manual` — состояние виртуальной машины управляется вручную;
- `AlwaysOnUnlessStoppedManually` — виртуальную машину можно остановить вручную (например, для обслуживания), но она автоматически запустится после перезагрузки хоста.

По умолчанию: `AlwaysOnUnlessStoppedManually`Допустимые значения: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

Пример:

```
runPolicy: AlwaysOnUnlessStoppedManually
```

■ **spec.virtualMachine.virtualMachineClassName**

строка

Обязательный параметр

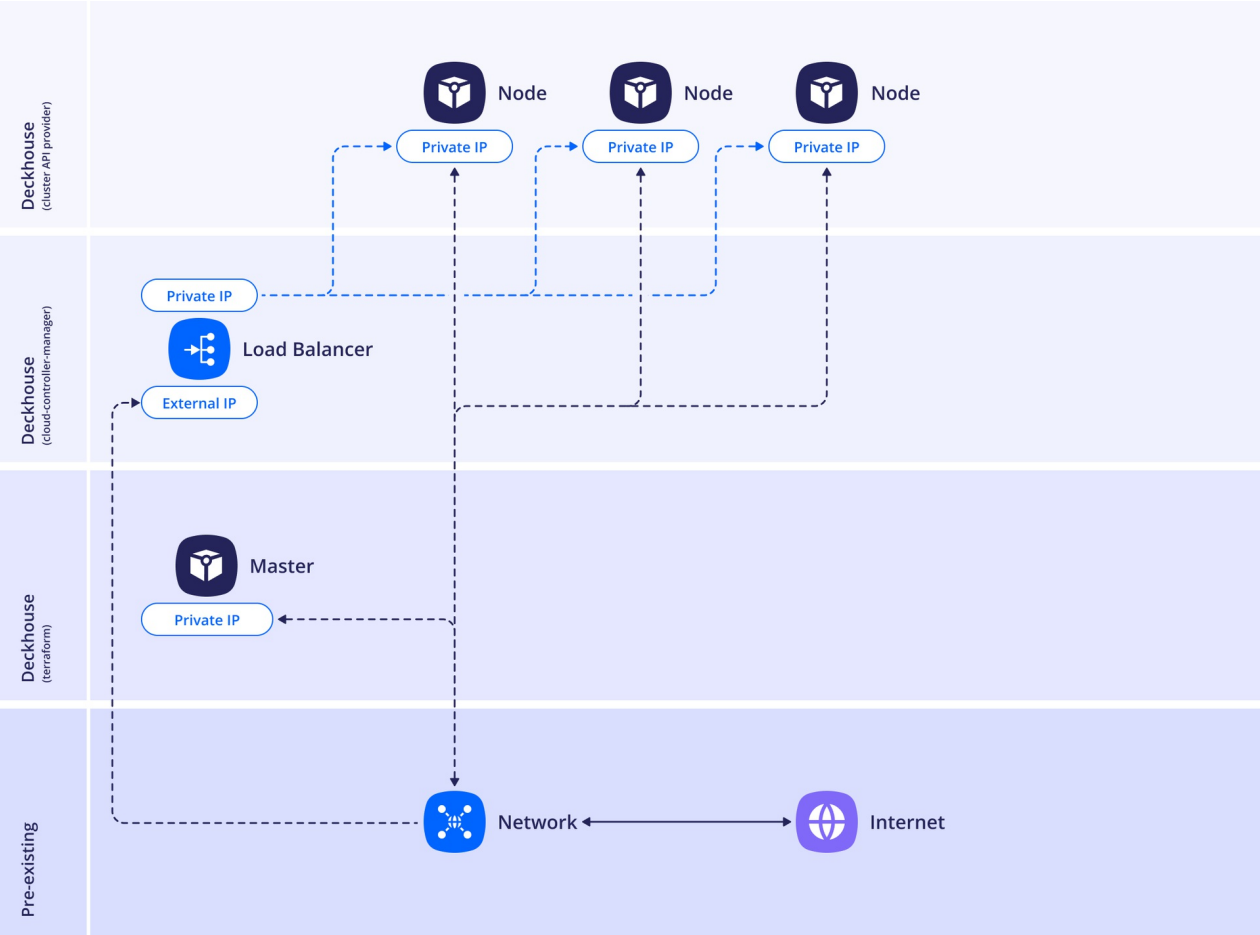
Имя `VirtualMachineClass`.

Ресурс `VirtualMachineClass` предназначен для централизованной конфигурации предпочтительных параметров виртуальных машин. Он позволяет задавать инструкции CPU, политики конфигурации

ресурсов CPU и памяти для виртуальных машин, а также устанавливать соотношения этих ресурсов.

Схемы размещения

Standard



Пример конфигурации схемы размещения:

```
---
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
masterNodeGroup:
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      ipAddresses:
        - Auto
      virtualMachineClassName: generic
  rootDisk:
    size: 50Gi
    storageClass: ceph-pool-r2-csi-rbd-immediate
  image:
    kind: ClusterVirtualImage
    name: ubuntu-2204
  etcdDisk:
    size: 15Gi
    storageClass: ceph-pool-r2-csi-rbd-immediate
provider:
  kubeconfigDataBase64: <KUBE_CONFIG>
  namespace: demo
```


Cloud provider — Basis Dynamix



Стадия жизненного цикла модуля: Preview

Модуль `cloud-provider-dynamix` обеспечивает интеграцию Deckhouse Kubernetes Platform с платформой Basis Dynamix. Он предоставляет возможность модулю `node-manager` использовать ресурсы Dynamix при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-dynamix`:

- Управление ресурсами Dynamix через `cloud-controller-manager`:
 - актуализирует метаданные виртуальных машин и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в Dynamix;
 - создаёт балансировщики нагрузки для Service с типом LoadBalancer. Для этого на Service нужно указать аннотации с именами внутренней и внешней сети.
- Заказ дисков через CSI-драйвер Dynamix (`dynamix.deckhouse.io`), чтобы из кластера можно было заказывать PersistentVolume.
- Заказ CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-decor`.
- Заказ CloudEphemeral-узлов через Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе `DynamixInstanceClass`.
- Регистрация в модуле `node-manager`, чтобы `DynamixInstanceClass` можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.



Модуль находится на стадии Preview.

Настройки



Стадия жизненного цикла модуля: Preview

Модуль автоматически включается для всех облачных кластеров, развернутых в Basis Dynamix.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dynamix`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left, there's a sidebar with navigation links: Обзор, Обновления, Модули (selected), Глобальные настройки, and Kubernetes. The main content area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. It contains a table of modules:

Имя	Состояние	Этап
admission-policy-engine	Включен	Генерация
backup	Выключен	—
basic-auth	Выключен	Генерация

On the right, the 'admission-policy-engine' module configuration is shown. It has a toggle switch labeled 'Модуль включен' which is turned on. Below it, there's a link to 'Документация к настройкам модуля admission-policy-engine'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-dynamix`:

```
d8 system module enable cloud-provider-dynamix
```

- **С помощью ModuleConfig `cloud-provider-dynamix`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-dynamix` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-dynamix`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dynamix
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dynamix`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект `ModuleConfig cloud-provider-dynamix` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-dynamix`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig cloud-provider-dynamix` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-dynamix
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig cloud-provider-dynamix`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig cloud-provider-dynamix`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dynamix
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Модуль не имеет настроек.

Настройки провайдера



Стадия жизненного цикла модуля: Preview



Если control plane кластера размещен на виртуальных машинах или bare-metal-серверах, cloud-провайдер использует настройки модуля `cloud-provider-dynamix` в конфигурации Deckhouse. Если control plane кластера размещен в облаке, cloud-провайдер использует структуру `DynamixClusterConfiguration` для настройки.

DynamixClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Базис.Dynamix.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```



После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: ssh-rsa AAAA
location: dynamix
account: acc_user
nodeNetworkCIDR: 10.241.32.0/24
nameservers:
- 10.0.0.10
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

- **account**
строка

Обязательный параметр

Имя аккаунта.
- **apiVersion**
строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`

- **kind**
строка
Обязательный параметр
Допустимые значения: `DynamixClusterConfiguration`
- **layout**
строка
Обязательный параметр
Название схемы размещения. Подробнее о возможных схемах размещения провайдера.
- **location**
строка
Обязательный параметр
Имя места установки кластера.
- **masterNodeGroup**
объект
Обязательный параметр
Спецификация для описания NodeGroup master-узлов.

i Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**
объект
Обязательный параметр
Частичное содержимое полей `DynamixInstanceClass`.
 - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
целочисленный
Размер диска для хранения данных etcd в GiB.
По умолчанию: `15`
Пример:

```
etcdDiskSizeGb: 15
```
 - **masterNodeGroup.instanceClass.externalNetwork**
строка
Обязательный параметр
Имя внешней сети.
Пример:

```
externalNetwork: extnet_vlan_1700
```
 - **masterNodeGroup.instanceClass.imageName**
строка
Обязательный параметр
Имя образа ОС, используемое для развертывания VM.
Пример:

```
imageName: '1'
```
 - **masterNodeGroup.instanceClass.memory**
целочисленный
Обязательный параметр
Память в MiB для выделения виртуальным машинам.
Допустимые значения: `1 <= X`
Пример:

```
memory: 8192
```
 - **masterNodeGroup.instanceClass.numCPUs**
целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
numCPUs: 2
```

- **masterNodeGroup.instanceClass.pool**

строка

Обязательный параметр

Имя пула хранения.

Пример:

```
pool: pool_a
```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**

целочисленный

Размер корневого диска в GiB для использования в виртуальных машинах.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

- **masterNodeGroup.instanceClass.storageEndpoint**

строка

Обязательный параметр

Имя хранилища.

Пример:

```
storageEndpoint: SharedTatlin_G1_SEP
```

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **nameservers**

массив строк

Список IP-адресов DNS-серверов.

По умолчанию: `[]`

- **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей DynamixInstanceClass.

- **nodeGroups.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для хранения данных etcd в GiB.

По умолчанию: `15`

Пример:

```
etcdDiskSizeGb: 15
```

- **nodeGroups.instanceClass.externalNetwork**

строка

Обязательный параметр

Имя внешней сети.

Пример:

```
externalNetwork: extnet_vlan_1700
```

■ **nodeGroups.instanceClass.imageName**

строка

Обязательный параметр

Имя образа ОС, используемое для развертывания VM.

Пример:

```
imageName: '1'
```

■ **nodeGroups.instanceClass.memory**

целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
memory: 8192
```

■ **nodeGroups.instanceClass.numCPUs**

целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
numCPUs: 2
```

■ **nodeGroups.instanceClass.pool**

строка

Имя пула хранения.

Пример:

```
pool: pool_a
```

■ **nodeGroups.instanceClass.rootDiskSizeGb**

целочисленный

Размер корневого диска в GiB для использования в виртуальных машинах.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

■ **nodeGroups.instanceClass.storageEndpoint**

строка

Имя хранилища.

Пример:

```
storageEndpoint: SharedTatlin_G1_SEP
```

○ **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

○ **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

■ **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```

annotations:
  ai.fleet.com/discombobulate: 'true'

```

■ **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`. Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```

labels:
  environment: production
  app: warp-drive-ai

```

■ **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```

taints:
- effect: NoExecute
  key: ship-class
  value: frigate

```

■ **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

■ **nodeGroups.nodeTemplate.taints.key**

строка

■ **nodeGroups.nodeTemplate.taints.value**

строка

○ **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

● **nodeNetworkCIDR**

строка

IP CIDR, который будет назначен внутреннему сегменту этой VINS.

● **provider**

объект

Обязательный параметр

Параметры для подключения к Базис.DynamiX.

○ **provider.appId**

строка

Обязательный параметр

ID приложения.

○ **provider.appSecret**

строка

Обязательный параметр

Секрет приложения.

○ **provider.controllerUrl**

строка

Обязательный параметр

URL-адрес конечной точки API Базис.DynamiX.

○ **provider.insecure**

булевый

Установите значение `true`, если Базис.DynamiX имеет самоподписанный сертификат.

По умолчанию: `false`

- **provider.oAuth2Url**
строка

Обязательный параметр

URL-адрес конечной точки SSO Базис.Dynamix API.

- **sshPublicKey**
строка

Обязательный параметр

Публичный ключ для доступа на узлы.

Подготовка окружения.



Стадия жизненного цикла модуля: Preview

Подготовка образа операционной системы

Вендоры операционных систем предоставляют специальные образы ОС (сборки), подготовленные для работы в облачной среде. Эти образы, как правило, содержат драйверы для виртуального оборудования, пакет `cloud-init`, гостевые агенты систем виртуализации и распространяются в виде образов дисков в форматах IMG или QCOW2. Мы рекомендуем использовать именно такие образы в качестве ОС на узлах ваших кластеров.

Образ операционной системы должен быть размещен в разделе «Образы» → «Шаблонные образы» на портале Basis Dynamix.

В случае, если в инфраструктуре отсутствует DNS-сервер, доступ к portalу Basis Dynamix из кластера можно организовать добавлением IP-адреса и доменов, связанных с порталом Basis Dynamix, в шаблон `cloud-init` для генерации файла `hosts`. Данный шаблон расположен в папке `/etc/cloud/templates/`. Название шаблона зависит от ОС.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.



После добавления данных в файл `hosts` и перед созданием шаблона из виртуальной машины, выполните команду `cloud-init clean`.

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.



Стадия жизненного цикла модуля: Preview

DynamixInstanceClass

Scope: Cluster

Version: v1

Параметры группы серверов Базис.Dynamix, которые будет использовать `CAPI` (модуль `node-manager`). На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.externalNetwork**
строка

Имя внешней сети.

Пример:

```
externalNetwork: extnet_vlan_1700
```

- **spec.imageName**
строка

Обязательный параметр

Имя образа ОС, используемое для развертывания VM.

Пример:

```
imageName: alt-p10-cloud-x86_64.img
```

- **spec.memory**

целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам.

Пример:

```
memory: 8192
```

- **spec.numCPUs**

целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам.

Пример:

```
numCPUs: 2
```

- **spec.rootDiskSizeGb**

целочисленный

Размер корневого диска в GiB для использования в виртуальных машинах.

По умолчанию: **30**

Пример:

```
rootDiskSizeGb: 30
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

FAQ



Стадия жизненного цикла модуля: Preview

Как настроить LoadBalancer?

Для настройки Service типа LoadBalancer добавьте в манифест Service следующие аннотации:

```
metadata:
  annotations:
    dynamix.cpi.flant.com/internal-network-name: <internal_name>
    dynamix.cpi.flant.com/external-network-name: <external_name>
```

Обе аннотации обязательны:

- `dynamix.cpi.flant.com/internal-network-name` — имя внутренней сети в Basis Dynamix;
- `dynamix.cpi.flant.com/external-network-name` — имя внешней сети в Basis Dynamix.

Термины «внутренняя сеть» и «внешняя сеть» используются в контексте Basis Dynamix. Внешняя сеть не обязательно должна быть публичной и может использовать серые IP-адреса.

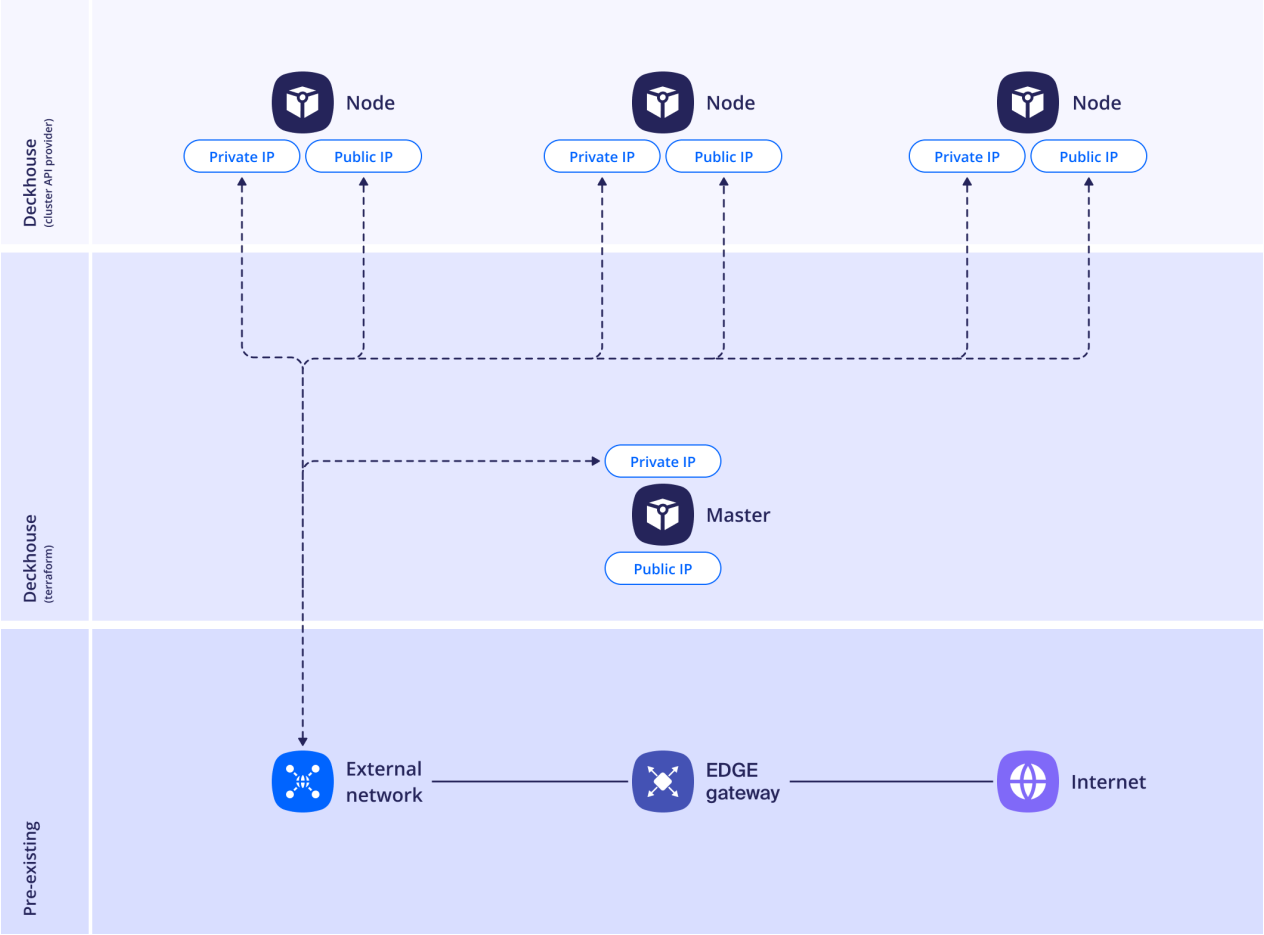
Если одна из аннотаций не указана, cloud-controller-manager завершит обработку Service с ошибкой.

Cloud provider - Basis Dynamix: схемы размещения



Стадия жизненного цикла модуля: Preview

Standard



Пример конфигурации схемы размещения:

```
---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"
```

StandardWithInternalNetwork

Пример конфигурации схемы размещения:

```

---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
nodeNetworkCIDR: "10.241.32.0/24"
nameservers: ["10.0.0.10"]
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"

```

Cloud provider — GCP

Модуль `cloud-provider-gcp` обеспечивает интеграцию Deckhouse Kubernetes Platform с Google Cloud Platform (GCP). Он предоставляет возможность модулю `node-manager` использовать ресурсы GCP при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-gcp`:

- Управление ресурсами GCP через `cloud-controller-manager`:
 - создаёт сетевые маршруты для сети PodNetwork на стороне GCP;
 - создаёт балансировщики нагрузки для Service с типом LoadBalancer;
 - актуализирует метаданные узлов кластера и удаляет из Kubernetes узлы, которых больше нет в GCP.
- Заказ дисков через CSI-драйвер Persistent Disk (`pd.csi.storage.gke.io`) и создание StorageClass для типов дисков GCP, чтобы из кластера можно было заказывать PersistentVolume.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-google`.
- Заказ CloudEphemeral-узлов через Machine Controller Manager (MCM). Параметры виртуальных машин задаются в ресурсе GCPIstanceClass.
- Регистрация в модуле `node-manager`, чтобы GCPIstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.

Настройки

Модуль настраивается автоматически, исходя из выбранной схемы размещения, определяемой в параметрах структуры GCPClusterConfiguration. В большинстве случаев нет необходимости ручной конфигурации модуля.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-gcp`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left, the sidebar has 'Модули' (Modules) selected. The main area shows a list of modules:

Имя	Состояние	Этап
admission-policy-engine	ВКЛЮЧЕН	Gen...
backup	ВЫКЛЮЧЕН	—
basic-auth	ВЫКЛЮЧЕН	Gen...

On the right, the 'admission-policy-engine' module configuration is shown, with a toggle switch for 'Модуль включен' (Module enabled) set to 'ON'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-gcp`:

```
d8 system module enable cloud-provider-gcp
```

- **С помощью ModuleConfig `cloud-provider-gcp`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-gcp` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-gcp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-gcp
spec:
  enabled: true
```

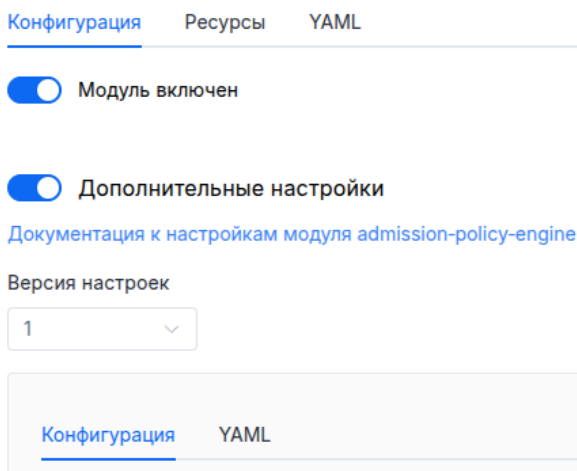
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-gcp`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `cloud-provider-gcp` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-gcp`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-gcp` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-gcp
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-gcp`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-gcp`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-gcp
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Количество и параметры процесса заказа машин в облаке настраиваются в custom resource `NodeGroup` модуля `node-manager`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference NodeGroup`). Инстанс-класс для cloud-провайдера GCP — это custom resource `GCPInstanceClass`, в котором указываются конкретные параметры самих машин.

Модуль автоматически создает `StorageClass`’ы, покрывающие все варианты дисков в GCP:

Тип	Репликация	Имя <code>StorageClass</code>
standard	none	pd-standard-not-replicated
standard	regional	pd-standard-replicated
balanced	none	pd-balanced-not-replicated
balanced	regional	pd-balanced-replicated
ssd	none	pd-ssd-not-replicated
ssd	regional	pd-ssd-replicated

Также он позволяет отфильтровать ненужные `StorageClass`’ы, для этого нужно указать их в параметре `exclude`.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.storageClass**

объект

- **settings.storageClass.default**

Параметр устарел
строка

Имя `StorageClass`, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический `StorageClass` по умолчанию будет определен следующим образом:

- присутствующий `StorageClass` в кластере, имеющий `default`-аннотацию;
- первый (по алфавиту) `StorageClass` из тех, что создаются модулем (в порядке из таблицы выше).



Параметр устарел. Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: pd-ssd-not-replicated
```

- **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) `StorageClass`, которые не будут созданы в кластере.

Пример:

```
exclude:
- pd-standard.*
- pd-ssd-replicated
```

Настройки провайдера

GCPClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в GCP.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
    - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
    - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
    additionalNetworkTags:
      - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**

строка

Обязательный параметр

Допустимые значения: `GCPClusterConfiguration`

- **labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (которые это поддерживают).

Если поменять теги в рабочем кластере, после конвержа необходимо пересоздать все машины, чтобы теги применились.

Подробнее про лейблы можно прочитать в официальной документации.

Формат — `key: value`.

- **layout**

строка

Обязательный параметр

Название схемы размещения.

`Standard` — включает режим Cloud NAT. Подробнее....

`WithoutNAT` — выделенная VPN создается для кластера. Все узлы имеют публичные IP-адреса. Подробнее....

Допустимые значения: `Standard`, `WithoutNAT`

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

◦ **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей GCPIstanceClass.

■ **masterNodeGroup.instanceClass.additionalLabels**

объект

Список дополнительных лейблов.

Подробнее про лейблы можно прочитать в документации Google Cloud.

Пример:

```
project: cms-production
severity: critical
```

■ **masterNodeGroup.instanceClass.additionalNetworkTags**

массив строк

Список дополнительных сетевых тегов (network tags).

Сетевые теги позволяют применять к инстансам firewall-правила. Подробнее о сетевых тегах можно прочитать в документации Google Cloud.

■ **masterNodeGroup.instanceClass.disableExternalIP**

булевый

Устанавливает, запретить назначение внешнего IP для инстанса или нет.

Параметр доступен только для layout `Standard`.

`true` — узлы не имеют публичных адресов, доступ в интернет осуществляется через `CloudNAT`.

`false` — для узлов создаются статические публичные адреса, они же используются для One-to-One NAT.

По умолчанию: `true`

Допустимые значения: `true`, `false`

■ **masterNodeGroup.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

Пример:

```
diskSizeGb: 50
```

■ **masterNodeGroup.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для etcd. Значение указывается в Гиб.

По умолчанию: `20`

Пример:

```
etcdDiskSizeGb: 20
```

■ **masterNodeGroup.instanceClass.image**

строка

Обязательный параметр

Образ, который будет использоваться в заказанных инстансах.

Список доступных образов можно найти в документации провайдера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации Deckhouse (учитывайте используемую версию Deckhouse).

Пример:

```
image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

■ **masterNodeGroup.instanceClass.machineType**

строка

Обязательный параметр

Тип заказываемых инстансов.

Внимание. Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

GCP позволяет указывать нестандартное количество CPU и RAM, например: `custom-8-40960` или `n2-custom-8-40960`.

Пример:

```
machineType: n1-standard-4
```

◦ **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов.

Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

◦ **masterNodeGroup.zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

• **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных фронтов или шлюзов).

◦ **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей GCPIstanceClass.

■ **nodeGroups.instanceClass.additionalLabels**

объект

Список дополнительных лейблов.

Подробнее про лейблы можно прочитать в документации Google Cloud.

Пример:

```
project: cms-production
severity: critical
```

■ **nodeGroups.instanceClass.additionalNetworkTags**

массив строк

Список дополнительных сетевых тегов (network tags).

Сетевые теги позволяют применять к инстансам firewall-правила. Подробнее о сетевых тегах можно прочитать в документации Google Cloud.

■ **nodeGroups.instanceClass.disableExternalIP**

булевый

Устанавливает, запретить назначение внешнего IP для инстанса или нет.

Параметр доступен только для layout `Standard`.

`true` — узлы не имеют публичных адресов, доступ в интернет осуществляется через `CloudNAT`.

`false` — для узлов создаются статические публичные адреса, они же используются для One-to-One NAT.

По умолчанию: `true`

Допустимые значения: `true`, `false`

■ **nodeGroups.instanceClass.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в `Гиб`.

Пример:

```
diskSizeGb: 50
```

■ **nodeGroups.instanceClass.image**

строка

Обязательный параметр

Образ, который будет использоваться в заказанных инстансах.

Список доступных образов можно найти в документации провайдера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации Deckhouse (учитывайте используемую версию Deckhouse).

Пример:

```
image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

■ **nodeGroups.instanceClass.machineType**

строка

Обязательный параметр

Тип заказываемых инстансов.

i **Внимание.** Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

GCP позволяет указывать нестандартное количество CPU и RAM, например: `custom-8-40960` или `n2-custom-8-40960`.

Пример:

```
machineType: n1-standard-4
```

○ **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, будет использоваться для генерации имен узлов.

○ **nodeGroups.nodeTemplate**

объект

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

■ **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

■ **nodeGroups.nodeTemplate.labels**

объект

Аналогично стандартному полю `metadata.labels`.

■ **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

■ **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

■ **nodeGroups.nodeTemplate.taints.key**

строка

■ **nodeGroups.nodeTemplate.taints.value**

строка

○ **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

○ **nodeGroups.zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

- **peeredVPCs**

массив строк

Список GCP VPC networks, с которыми будет объединена сеть кластера.

Сервис-аккаунт должен иметь доступ ко всем перечисленным VPC. Если доступа нет, пиринг необходимо настраивать вручную.

- **provider**

объект

Обязательный параметр

Параметры подключения к API GCP.

- **provider.region**

строка

Обязательный параметр

Имя региона, в котором будут заказываться инстансы.

- **provider.serviceAccountJSON**

строка

Обязательный параметр

Ключ к Service Account'у с правами Project Admin (`service account key`) в JSON-формате.

Как получить `service account key`.

- **sshAllowList**

массив строк

Список CIDR, разрешенных для подключения к узлам по SSH.

По умолчанию — с любого адреса.

- **sshKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы под пользователем `user`.

- **standard**

объект

Настройки для схемы размещения `Standard`.

- **standard.cloudNATAddresses**

массив строк

Список имен публичных статических IP-адресов для `Cloud NAT`.

- **subnetworkCIDR**

строка

Подсеть, в которой будут работать узлы кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withoutNAT**

объект

Настройки для схемы размещения `WithoutNAT`.

- **zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

Подготовка окружения



Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

Чтобы Deckhouse Kubernetes Platform могла управлять ресурсами, в Google Cloud необходимо создать service account. Далее представлена краткая последовательность действий по созданию service account. Если вам необходима более подробная инструкция, вы можете найти ее в документации провайдера.



Внимание! Созданный service account key невозможно восстановить, только удалить и создать новый.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Настройка через Google Cloud Console

Перейдите по ссылке, выберите проект и создайте новый service account (также можно выбрать уже существующий).

Созданному service account'у должны быть присвоены несколько необходимых ролей:

```
Compute Admin
Service Account User
Network Management Admin
```

Роли можно присвоить на этапе создания service account'а либо изменить на этой странице.

Чтобы получить service account key в JSON-формате, на странице в колонке Actions нажмите на три вертикальные точки и выберите `Manage keys`. Затем нажмите `Add key -> Create new key -> Key type -> JSON`.

Настройка через gcloud CLI

Для настройки через интерфейс командной строки выполните следующие шаги:

1. Экспортируйте переменные окружения:

```
export PROJECT_ID=sandbox
export SERVICE_ACCOUNT_NAME=deckhouse
```

2. Выберите project:

```
gcloud config set project $PROJECT_ID
```

3. Создайте service account:

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

4. Присвойте роли созданному service account:

```
for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
--role=${role}; done
```

Список необходимых ролей:

```
roles/compute.admin
roles/iam.serviceAccountUser
roles/networkmanagement.admin
```

5. Выполните проверку ролей service account:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
--filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Создайте service account key:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

GCPInstanceClass

Scope: Cluster

- v1
- v1alpha1

Описывает параметры группы GCP Instances, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalDisks**
массив объектов

Список дополнительных дисков, которые будут подключены к инстансу.

Полезно для нагрузок, связанных с хранением данных, таких как LINSTOR, Ceph, NFS и аналогичные решения.

Пример:

```
additionalDisks:
- size: 100
  type: pd-ssd
```

- **spec.additionalDisks.autoDelete**
булевый

Автоматически удалять диск при удалении инстанса.

По умолчанию: `true`

- **spec.additionalDisks.provisionedIops**
целочисленный

Запрошенное количество IOPS для диска.

Обязательно для типов `pd-extreme` и `hyperdisk-extreme`.

Допустимые значения: `1 <= X`

- **spec.additionalDisks.provisionedThroughput**
целочисленный

Запрошенная пропускная способность диска в МиБ/с.

Применимо для типов `hyperdisk-balanced` и `hyperdisk-throughput`.

Допустимые значения: `1 <= X`

- **spec.additionalDisks.size**
целочисленный

Обязательный параметр

Размер диска в гигабайтах (ГБ).

Допустимые значения: `1 <= X`

Пример:

```
size: 100
```

■ **spec.additionalDisks.type**

строка

Обязательный параметр

Тип диска.

- `pd-standard` — стандартный Persistent Disk (HDD).
- `pd-balanced` — сбалансированный Persistent Disk (SSD).
- `pd-ssd` — SSD Persistent Disk.
- `pd-extreme` — экстремальный Persistent Disk; требует указания `provisionedIops`.
- `hyperdisk-balanced` — Hyperdisk со сбалансированной производительностью.
- `hyperdisk-throughput` — Hyperdisk, оптимизированный для пропускной способности.
- `hyperdisk-extreme` — Hyperdisk с максимальной производительностью; требует указания `provisionedIops`.

Допустимые значения: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

Пример:

```
type: pd-ssd
```

○ **spec.additionalLabels**

объект

Список дополнительных лейблов.

Подробно про лейблы можно прочитать в официальной документации.

Пример:

```
additionalLabels:
  project: cms-production
  severity: critical
```

○ **spec.additionalNetworkTags**

массив строк

Список дополнительных тегов.

Теги, например, позволяют применять к экземплярам правила firewall. Подробно про network tags можно прочитать в официальной документации.

○ **spec.capacity**

Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа экземпляра Deckhouse использует параметры, полученные из облака.

Емкость экземпляра для *нестандартных* типов экземпляров (параметр `machineType`).

Используется cluster-autoscaler'ом при планировании, только когда в NodeGroup'е еще нет узлов (при `minPerZone` равном 0). Если в NodeGroup уже есть узлы, cluster-autoscaler использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов экземпляров. Параметры *стандартных* типов экземпляров (`a2-highgpu-8g`, `c2-standard-8` и т. п.) заложены в Deckhouse.

■ **spec.capacity.cpu**

строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

■ **spec.capacity.memory**

строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.disableExternalIP**

булевый

Запрет назначения внешнего IP-адреса для инстанса.

- **spec.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в Гиб.

По умолчанию: 50

Пример:

```
diskSizeGb: 20
```

- **spec.diskType**

строка

Тип созданного диска.

По умолчанию: pd-standard

Допустимые значения: pd-standard, pd-ssd

Пример:

```
diskType: pd-ssd
```

- **spec.enableNestedVirtualization**

булевый

Включить вложенную виртуализацию на инстансе.

Необходимо для запуска виртуальных машин (например, KVM) внутри GCP-инстансов.

Внимание. Поддерживается только на определённых типах машин. Список совместимых типов приведён в документации GCP.

По умолчанию: false

- **spec.image**

строка

Необязательный параметр.

Образ, который будет использоваться в заказанных инстансах.

Список доступных образов можно найти в документации провайдера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации Deckhouse (учитывайте используемую версию Deckhouse).

Пример:

```
image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

- **spec.machineType**

строка

Обязательный параметр

Тип заказываемых инстансов.

Внимание! Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре zones.

GCP позволяет указывать нестандартное количество CPU и RAM, например: custom-8-40960 или n2-custom-8-40960.

Пример:

```
machineType: n1-standard-4
```

- **spec.preemptible**

булевый

Необходимость заказа preemptible-инстансов (прерываемых виртуальных машин).

По умолчанию: false

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

Описывает параметры группы GCP Instances, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalDisks**
массив объектов

Список дополнительных дисков, которые будут подключены к экземпляру.

Полезно для нагрузок, связанных с хранением данных, таких как LINSTOR, Ceph, NFS и аналогичные решения.

Пример:

```
additionalDisks:
- size: 100
  type: pd-ssd
```

- **spec.additionalDisks.autoDelete**
булевый

Автоматически удалять диск при удалении экземпляра.

По умолчанию: `true`

- **spec.additionalDisks.provisionedIops**
целочисленный

Запрошенное количество IOPS для диска.

Обязательно для типов `pd-extreme` и `hyperdisk-extreme`.

Допустимые значения: `1 <= X`

- **spec.additionalDisks.provisionedThroughput**
целочисленный

Запрошенная пропускная способность диска в МБ/с.

Применимо для типов `hyperdisk-balanced` и `hyperdisk-throughput`.

Допустимые значения: `1 <= X`

- **spec.additionalDisks.size**
целочисленный

Обязательный параметр

Размер диска в гигабайтах (ГБ).

Допустимые значения: `1 <= X`

Пример:

```
size: 100
```

- **spec.additionalDisks.type**
строка

Обязательный параметр

Тип диска.

- `pd-standard` — стандартный Persistent Disk (HDD).
- `pd-balanced` — сбалансированный Persistent Disk (SSD).
- `pd-ssd` — SSD Persistent Disk.
- `pd-extreme` — экстремальный Persistent Disk; требует указания `provisionedIops`.
- `hyperdisk-balanced` — Hyperdisk со сбалансированной производительностью.
- `hyperdisk-throughput` — Hyperdisk, оптимизированный для пропускной способности.
- `hyperdisk-extreme` — Hyperdisk с максимальной производительностью; требует указания `provisionedIops`.

Допустимые значения: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

Пример:

```
type: pd-ssd
```

- **spec.additionalLabels**
объект

Список дополнительных лейблов.

Подробнее про лейблы можно прочитать в официальной документации.

Пример:

```
additionalLabels:
  project: cms-production
  severity: critical
```

- **spec.additionalNetworkTags**

массив строк

Список дополнительных тегов.

Теги, например, позволяют применять к инстансам правила firewall. Подробнее про network tags можно прочитать в официальной документации.

- **spec.capacity**

Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `machineType`).

Используется cluster-autoscaler'ом при планировании, только когда в NodeGroup'е еще нет узлов (при `minPerZone` равно 0). Если в NodeGroup уже есть узлы, cluster-autoscaler использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`a2-highgpu-8g`, `c2-standard-8` и т. п.) заложены в Deckhouse.

- **spec.capacity.cpu**

строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

- **spec.capacity.memory**

строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

- **spec.disableExternalIP**

булевый

Запрет назначения внешнего IP-адреса для инстанса.

- **spec.diskSizeGb**

целочисленный

Размер root-диска. Значение указывается в **Гиб**.

По умолчанию: `50`

Пример:

```
diskSizeGb: 20
```

- **spec.diskType**

строка

Тип созданного диска.

По умолчанию: `pd-standard`

Допустимые значения: `pd-standard`, `pd-ssd`

Пример:

```
diskType: pd-ssd
```

- **spec.enableNestedVirtualization**

булевый

Включить вложенную виртуализацию на инстансе.

Необходимо для запуска виртуальных машин (например, KVM) внутри GCP-инстансов.

Внимание. Поддерживается только на определённых типах машин. Список совместимых типов приведён в документации GCP.

По умолчанию: `false`

- **spec.image**

строка

Необязательный параметр.

Образ, который будет использоваться в заказанных инстансах.

Список доступных образов можно найти в документации провайдера.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации Deckhouse (учитывайте используемую версию Deckhouse).

Пример:

```
image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

- **spec.machineType**

строка

Обязательный параметр

Тип заказываемых инстансов.

Внимание! Следует убедиться, что указанный тип есть во всех зонах, перечисленных в параметре `zones`.

GCP позволяет указывать нестандартное количество CPU и RAM, например: `custom-8-40960` или `n2-custom-8-40960`.

Пример:

```
machineType: n1-standard-4
```

- **spec.preemptible**

булевый

Необходимость заказа preemptible-инстансов (прерываемых виртуальных машин).

По умолчанию: `false`

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

FAQ

Как поднять кластер

1. Настройте облачное окружение.
2. Включите модуль или передайте флаг `--extra-config-map-data base64_encoding_of_custom_config` с параметрами модуля в скрипт установки `install.sh`.
3. Создайте один или несколько custom resource `GCPInstanceClass`.
4. Создайте один или несколько custom resource `NodeGroup` для управления количеством и процессом заказа машин в облаке.

Добавление CloudStatic узлов в кластер

К виртуальным машинам, которые вы хотите добавить к кластеру в качестве узлов, добавьте `Network Tag`, аналогичный префиксу кластера.

Префикс кластера можно узнать, воспользовавшись следующей командой:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
| base64 -d | grep prefix
```

Примеры

Пример кастомного ресурса `GCPInstanceClass`

Ниже представлен простой пример конфигурации кастомного ресурса `GCPInstanceClass`:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: test
spec:
  machineType: n1-standard-1
```

Включение вложенной виртуализации

Для запуска виртуальных машин (например, KVM) внутри GCP-инстансов необходимо включить вложенную виртуализацию.

i Вложенная виртуализация поддерживается только на определённых типах машин. Список совместимых типов приведён в документации GCP.

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  enableNestedVirtualization: true
```

Добавление дополнительных дисков

Чтобы подключить к инстансам дополнительные диски (например, для узлов хранилища LINSTOR, Ceph, NFS и аналогичных решений), задайте их в параметре `additionalDisks`:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
    - size: 200
      type: pd-ssd
    - size: 500
      type: pd-standard
    autoDelete: true
```

Настройка политик безопасности на узлах

На виртуальных машинах кластера в GCP может возникнуть необходимость ограничить или расширить входящий и исходящий трафик по различным причинам. Некоторые из них могут включать:

- Разрешение подключения к узлам кластера с виртуальных машин из другой подсети.
- Разрешение подключения к портам статического узла для работы приложения.
- Ограничение доступа к внешним ресурсам или другим виртуальным машинам в облаке по требованию службы безопасности.

Для всего этого необходимо применять дополнительные `network tags`.

Установка дополнительных `network tags` на статических и `master`-узлах

Данный параметр можно задать либо при создании кластера или в уже существующем кластере. В обоих случаях дополнительные `network tags` указываются в `GCPClusterConfiguration`:

- для `master`-узлов — в секции `masterNodeGroup` в поле `additionalNetworkTags`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей соответствующую `nodeGroup`, в поле `additionalNetworkTags`.

Поле `additionalNetworkTags` содержит массив строк с именами `network tags`.

Установка дополнительных `network tags` на эфемерных узлах

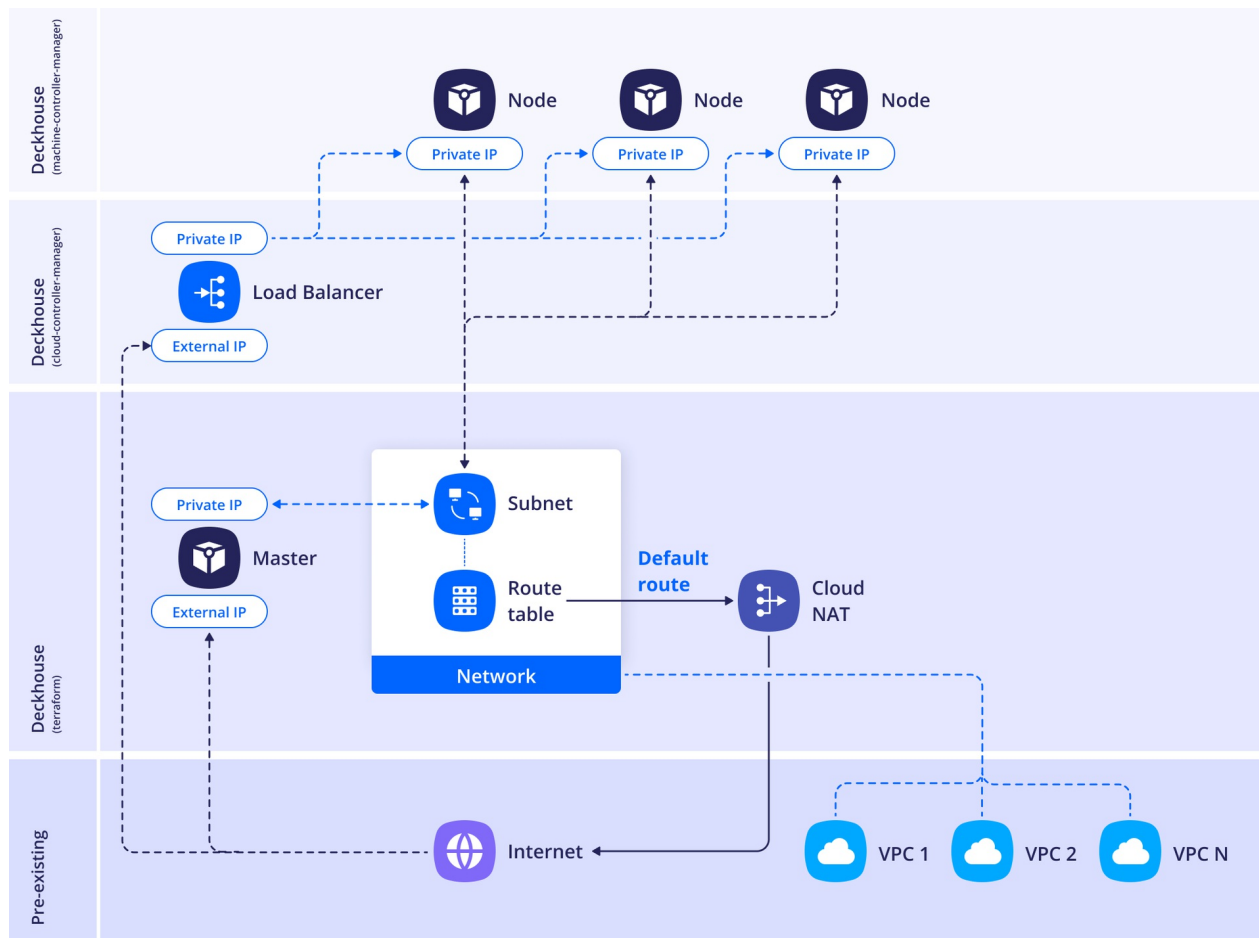
Необходимо указать параметр `additionalNetworkTags` для всех `GCPInstanceClass` в кластере, которым нужны дополнительные `network tags`.

Схемы размещения

Поддерживаются две схемы размещения. Ниже подробнее о каждой из них.

Standard

- Для кластера создается отдельная VPC с Cloud NAT.
- Узлы в кластере не имеют публичных IP-адресов.
- Публичные IP-адреса можно назначить на статические и master-узлы:
 - При этом будет использоваться One-to-One NAT для отображения публичного IP-адреса в IP-адрес узла (следует помнить, что CloudNAT в этом случае использоваться не будет).
- Если master не имеет публичного IP-адреса, для установки и доступа в кластер необходим дополнительный инстанс с публичным IP-адресом (например, bastion-хост).
- Между VPC кластера и другими VPC можно настроить пиринговое соединение.



Пример конфигурации схемы размещения:

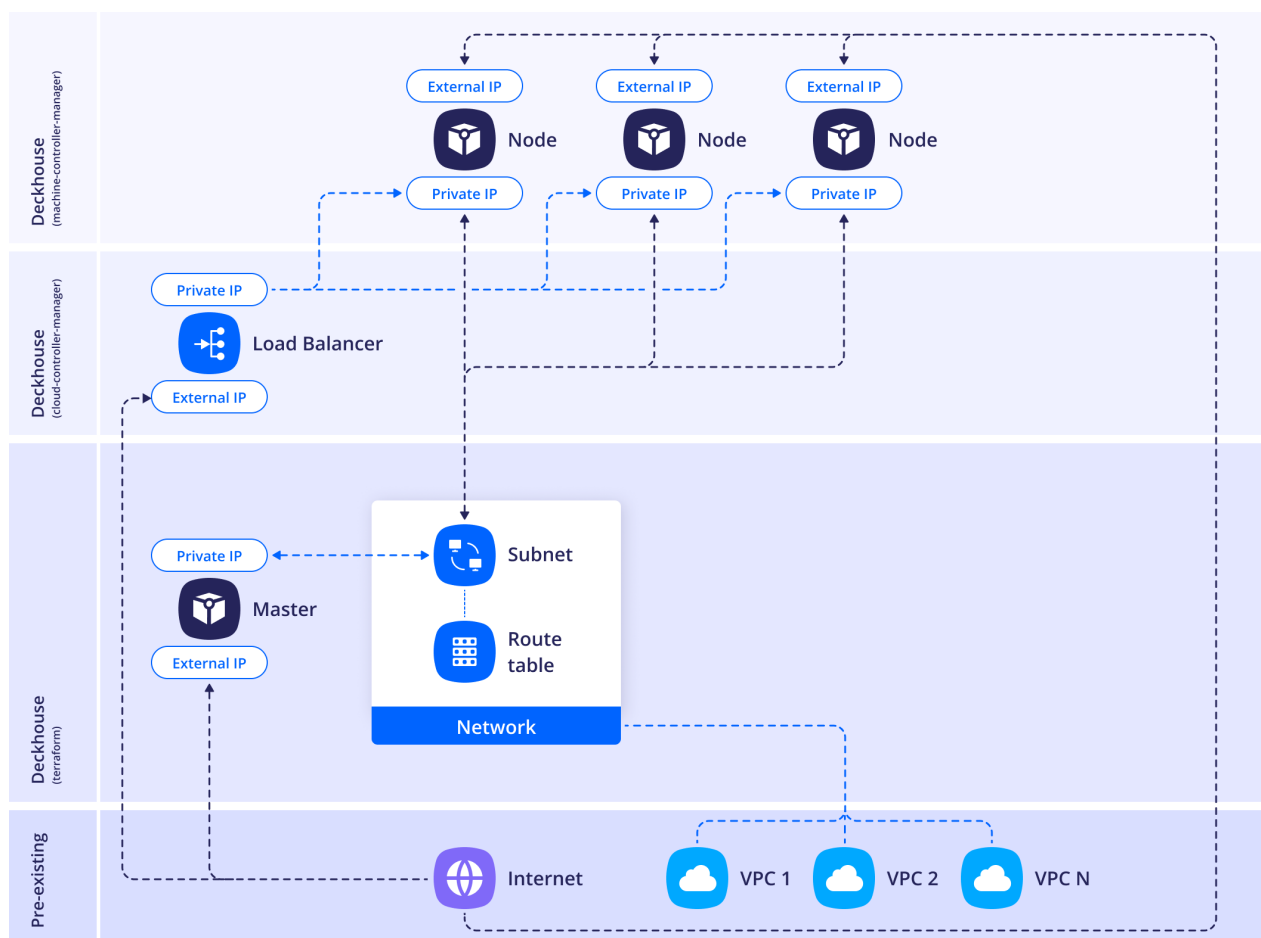
```

apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Необязательный параметр. Адреса из этого списка будут использованы для
  # адресации Cloud NAT.
  cloudNATAddresses:
    - example-address-1
    - example-address-2
  subnetworkCIDR: 10.0.0.0/24      # Обязательный параметр.
  # Необязательный параметр, список GCP VPC Networks, с которыми Kubernetes VPC
  # Network будет соединяться через пиринговое соединение.
  peeredVPCs:
    - default
  sshKey: "<SSH_PUBLIC_KEY>" # Обязательный параметр.
  labels:
    kube: example
  masterNodeGroup:
    replicas: 1
    zones:                                     # Необязательный параметр.
      - europe-west4-b
    instanceClass:
      machineType: n1-standard-4      # Обязательный параметр.
      image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
      diskSizeGb: 20                  # Необязательный параметр. Если не указан – используется локальный
диск.
      disableExternalIP: false        # Необязательный параметр, по умолчанию master-узел имеет
externalIP.
      additionalNetworkTags:          # Необязательный параметр.
        - tag1
      additionalLabels:               # Необязательный параметр.
        kube-node: master
  nodeGroups:
    - name: static
      replicas: 1
      zones:                                     # Необязательный параметр.
        - europe-west4-b
      instanceClass:
        machineType: n1-standard-4      # Обязательный параметр.
        image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
        diskSizeGb: 20                  # Необязательный параметр. Если не указан – используется локальный
диск.
        disableExternalIP: true        # Необязательный параметр, по умолчанию узлы не имеют externalIP.
        additionalNetworkTags:          # Необязательный параметр.
          - tag1
        additionalLabels:               # Необязательный параметр.
          kube-node: static
  provider:
    region: europe-west4                # Обязательный параметр.
    serviceAccountJSON: |              # Обязательный параметр.
      {
        "type": "service_account",
        "project_id": "sandbox",
        "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
        "private_key": "-----BEGIN PRIVATE KEY-----",
        "client_id": "342975658279248",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://oauth2.googleapis.com/token",
        "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
        "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
      }

```

WithoutNAT

- Для кластера создается отдельная VPC, все узлы кластера имеют публичные IP-адреса.
- Между VPC кластера и другими VPC можно настроить пиринговое соединение.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24      # Обязательный параметр.
# Необязательный параметр, список GCP VPC Networks, с которыми Kubernetes VPC
# Network будет соединяться через пиринговое соединение.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                                # Необязательный параметр.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4      # Обязательный параметр.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
    diskSizeGb: 20                  # Необязательный параметр, Если не указан — используется локальный
диск.
    additionalNetworkTags:          # Необязательный параметр.
    - tag1
    additionalLabels:               # Необязательный параметр.
    kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                                # Необязательный параметр.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4      # Обязательный параметр.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Обязательный
параметр.
    diskSizeGb: 20                  # Необязательный параметр. Если не указан — используется локальный
диск.
    additionalNetworkTags:          # Необязательный параметр.
    - tag1
    additionalLabels:               # Необязательный параметр.
    kube-node: static
provider:
  region: europe-west4              # Обязательный параметр.
  serviceAccountJSON: |             # Обязательный параметр.
  {
    "type": "service_account",
    "project_id": "sandbox",
    "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
    "private_key": "-----BEGIN PRIVATE KEY-----",
    "client_id": "342975658279248",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
  }

```

Cloud provider — Huawei Cloud

Модуль `cloud-provider-huaweicloud` обеспечивает интеграцию Deckhouse Kubernetes Platform с Huawei Cloud. Он предоставляет возможность модулю `node-manager` использовать ресурсы Huawei Cloud при заказе узлов для `NodeGroup`.

Функции модуля `cloud-provider-huaweicloud`:

- Управление ресурсами Huawei Cloud через `cloud-controller-manager`:
 - актуализирует метаданные инстансов и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в облаке;
 - создаёт балансировщики нагрузки (ELB) для Service с типом LoadBalancer.
- Заказ дисков через CSI-драйвер EVS (`evs.csi.huaweicloud.com`), чтобы из кластера можно было заказывать PersistentVolume.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-huaweicloud`.
- Заказ CloudEphemeral-узлов через Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе `HuaweiCloudInstanceClass`.
- Регистрация в модуле `node-manager`, чтобы `HuaweiCloudInstanceClass` можно было указывать при описании `NodeGroup`.

- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.

Настройки

Модуль автоматически включается для всех облачных кластеров, развернутых в Huawei Cloud.

В модуле реализован 1 алерт.

Модуль не имеет настроек.

Настройки провайдера

HuaweiCloudClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Huawei Cloud.

Используется облачным провайдером, если управляющий слой кластера размещён в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

Пример:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
standard:
  externalNetworkName: external-network
provider:
  domainName: "<DOMAIN_NAME>"
  region: eu-3
  accessKey: "<USERNAME>"
  secretKey: "<PASSWORD>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    flavorName: m1.large
    rootDiskSize: 50
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`

- **kind**

строка

Обязательный параметр

Допустимые значения: `HuaweiCloudClusterConfiguration`

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Возможные схемы размещения провайдера описаны на странице [Схемы размещения](#).

- **masterNodeGroup**

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей HuaweiCloudInstanceClass.

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для etcd. Значение указывается в гигабайтах.

По умолчанию: `10`

Пример:

```
etcdDiskSizeGb: 10
```

- **masterNodeGroup.instanceClass.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

- **masterNodeGroup.instanceClass.imageName**

строка

Обязательный параметр

Имя образа виртуальной машины, который будет использоваться для инстанса.

Список поддерживаемых Deckhouse ОС и их версий можно найти на странице Поддерживаемые версии Kubernetes и ОС (учитывайте используемую версию Deckhouse).

- **masterNodeGroup.instanceClass.rootDiskSize**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска.

По умолчанию: `50`

Пример:

```
rootDiskSize: 50
```

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **masterNodeGroup.serverGroup**

объект

ServerGroup объединяет инстансы в группу. Инстансы в такой группе будут размещены на одном гипервизоре (affinity) или на разных гипервизорах (anti-affinity). Это позволяет повысить отказоустойчивость кластера.

- **masterNodeGroup.serverGroup.policy**

строка

Обязательный параметр

Политика распределения инстансов:

- `AntiAffinity` — размещает инстансы на разных гипервизорах. Данной политикой управляет Deckhouse.

Допустимые значения: `AntiAffinity`

- **masterNodeGroup.volumeTypeMap**

объект

Обязательный параметр

Словарь, сопоставляющий зонам доступности типы дисков для worker-узлов группы.

Формат записи: `<ЗОНА ДОСТУПНОСТИ>: <ТИП ДИСКА>` (см. пример).

Если указан параметр `rootDiskSize`, тот же тип диска используется для загрузочного диска виртуальной машины.

Рекомендуется использовать самые быстрые типы дисков, доступные у провайдера.

Если свойство `volumeTypeMap` не задано для группы узлов, для этой группы в качестве значения по умолчанию используется `volumeTypeMap` из `masterNodeGroup`.

Если значение `replicas` превышает число записей в словаре, назначение типов выполняется по кругу (с начала словаря). Например, при `replicas: 5` для группы `front` и словаре, где первым указан `fast-eu-1a`, а вторым — `fast-eu-1b`, узлы `front-0`, `front-2`, `front-4` получают `fast-eu-1a`, а `front-1` и `front-3` — `fast-eu-1b`.

Пример:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей HuaweiCloudInstanceClass.

- **nodeGroups.instanceClass.additionalNetworks**

массив строк

Необязательный параметр.

Список ID сетей (подсетей) для дополнительных сетевых интерфейсов (NIC).

- **nodeGroups.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для etcd. Значение указывается в гигабайтах.

По умолчанию: `10`

Пример:

```
etcdDiskSizeGb: 10
```

- **nodeGroups.instanceClass.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

- **nodeGroups.instanceClass.imageName**

строка

Обязательный параметр

Имя образа виртуальной машины, который будет использоваться для инстанса.

Список поддерживаемых Deckhouse ОС и их версий можно найти на странице Поддерживаемые версии Kubernetes и ОС (учитывайте используемую версию Deckhouse).

- **nodeGroups.instanceClass.mainNetwork**

строка

Необязательный параметр.

ID основной сети для основного сетевого интерфейса (NIC) виртуальной машины.

По умолчанию используется значение из `HuaweiCloudDiscoveryData`.

- **nodeGroups.instanceClass.rootDiskSize**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска.

По умолчанию: `50`

Пример:

```
rootDiskSize: 50
```

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **nodeGroups.volumeTypeMap**

объект

Словарь типов дисков для рабочих узлов группы.

Формат элементов словаря: `<ЗОНА ДОСТУПНОСТИ>: <ТИП ДИСКА>` (см. пример).

Если указан параметр `rootDiskSize`, этот же тип диска будет использован в качестве загрузочного диска виртуальной машины.

Всегда рекомендуется использовать самые быстрые диски, предоставляемые провайдером.

Если свойство `volumeTypeMap` не указано для группы узлов, будет использовано значение `volumeTypeMap` из `masterNodeGroup` в качестве запасного варианта для этой группы узлов.

Если значение, указанное в `replicas`, превышает количество элементов в словаре, узлы, чьи номера превышают длину словаря, получают значения, начиная с начала словаря. Если для словаря из примера для группы `front` указано `replicas: 5`, с типом диска `fast-eu-1a` будут `front-0`, `front-2` и `front-4`, а с типом диска `fast-eu-1b` будут `front-1` и `front-3`.

Пример:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **provider**

объект

Обязательный параметр

Параметры подключения к API Huawei Cloud.

- **provider.accessKey**
строка
Обязательный параметр
Ключ доступа к облаку Huawei Cloud.
- **provider.authURL**
строка
URL-адрес аутентификации личности.
- **provider.cloud**
строка
Обязательный параметр
Конечная точка поставщика облачных услуг.
- **provider.domainName**
строка
Имя учетной записи IAM для охвата.
- **provider.enterpriseProjectID**
строка
Идентификатор подпроекта.
- **provider.insecure**
булевый
Доверять самоподписанным SSL-сертификатам.
- **provider.projectID**
строка
Идентификатор проекта.
- **provider.region**
строка
Обязательный параметр
Регион Huawei Cloud, в котором будет развернут кластер.
- **provider.secretKey**
строка
Обязательный параметр
Секретный ключ облака Huawei Cloud.

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **standard**

объект

Настройки для схемы размещения `Standard`.

- **standard.enableEIP**
булевый
Включите Elastic IP для мастер узлов.
Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`
- **standard.internalNetworkCIDR**

строка

Обязательный параметр

Адресация для внутренней сети узлов кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0-9))$`

- **standard.internalNetworkDNSServers**

массив строк

Список адресов рекурсивных DNS внутренней сети.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **standard.internalNetworkSecurity**

булевый

Флаг, который определяет необходимость настройки SecurityGroups на портах внутренней сети.

По умолчанию: `true`

- **vpcPeering**

объект

Настройки для схемы размещения VpcPeering.

- **vpcPeering.internalNetworkCIDR**

строка

Обязательный параметр

Адресация для внутренней сети узлов кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0-9))$`

- **vpcPeering.internalNetworkDNSServers**

массив строк

Список адресов рекурсивных DNS внутренней сети узлов кластера.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **vpcPeering.internalNetworkSecurity**

булевый

Флаг, который определяет необходимость настройки SecurityGroups на портах внутренней сети.

По умолчанию: `true`

- **vpcPeering.subnet**

строка

Имя подсети в сети VPC.

- **zones**

массив строк

Необязательный параметр.

Глобальное ограничение набора зон, с которыми работает данный облачный провайдер.

Подготовка окружения



Чтобы использовать ContainerdV2 в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка CgroupsV2;
- systemd версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для взаимодействия с ресурсами в облаке Huawei Cloud компоненты Deckhouse Kubernetes Platform используют API Huawei Cloud. Для настройки этого подключения требуется создать пользователя в сервисе Huawei Cloud IAM и назначить ему соответствующие права доступа.

Настройка IAM через веб-интерфейс

Для настройки IAM через веб-интерфейс сначала создайте новую группу пользователей и назначьте ей необходимые права. Для этого выполните следующие шаги:

1. Перейдите в раздел «Identity and Access Management (IAM)».
2. Откройте страницу «User Groups» и нажмите «Create User Group».
3. В поле «Name» укажите имя группы (например, `deckhouse`).
4. Нажмите «OK» для создания группы.
5. Выберите созданную группу из списка.
6. На вкладке «Permissions» нажмите «Authorize».
7. Укажите следующие политики: «ECS Admin», «VPC Administrator», «NAT Admin», «ELB FullAccess», «DEW KeypairFullAccess».
8. Нажмите «Next», затем «OK» и завершите настройку, нажав «Finish».

Добавьте нового пользователя. Для этого выполните следующие шаги:

1. Перейдите на страницу «Users» в разделе IAM и нажмите «Create User».
2. В поле «Username» введите имя пользователя (например, `deckhouse`).
3. Установите «Access type» в значение «Programmatic access» и убедитесь, что «Management console access» отключен.
4. Выберите «Access key» в качестве «Credential Type».
5. Нажмите «Next».
6. Выберите ранее созданную группу пользователей.
7. Нажмите «Create», чтобы завершить создание пользователя.
8. Нажмите «OK», чтобы загрузить Access Key ID и Secret Access Key. Убедитесь, что вы сохранили эти данные в надёжном месте, так как они понадобятся для доступа к API.

JSON-политики

Далее приведено содержание политик в формате JSON:

Политика «ECS Admin»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "ecs:*:*",
        "evs:*:get",
        "evs:*:list",
        "evs:volumes:create",
        "evs:volumes:delete",
        "evs:volumes:attach",
        "evs:volumes:detach",
        "evs:volumes:manage",
        "evs:volumes:update",
        "evs:volumes:use",
        "evs:volumes:uploadImage",
        "evs:snapshots:create",
        "vpc:*:get",
        "vpc:*:list",
        "vpc:networks:create",
        "vpc:networks:update",
        "vpc:subnets:update",
        "vpc:subnets:create",
        "vpc:ports:*",
        "vpc:routers:get",
        "vpc:routers:update",
        "vpc:securityGroups:*",
        "vpc:securityGroupRules:*",
        "vpc:floatingIps:*",
        "vpc:publicIps:*",
        "ims:images:create",
        "ims:images:delete",
        "ims:images:get",
        "ims:images:list",
        "ims:images:update",
        "ims:images:upload"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «VPC Administrator»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "vpc:vpcs:*",
        "vpc:routers:*",
        "vpc:networks:*",
        "vpc:subnets:*",
        "vpc:ports:*",
        "vpc:privateIps:*",
        "vpc:peerings:*",
        "vpc:routes:*",
        "vpc:lbaas:*",
        "vpc:vpns:*",
        "ecs:*:get",
        "ecs:*:list",
        "elb:*:get",
        "elb:*:list"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «NAT Admin»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "nat:*:*",
        "vpc:*:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «DEW KeypairFullAccess»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "kps:domainKeypairs:*",
        "ecs:serverKeypairs:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Политика «ELB FullAccess»

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "elb:*:*",
        "vpc:*:get*",
        "vpc:*:list*",
        "ecs:*:get*",
        "ecs:*:list*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

HuaweiCloudInstanceClass

Scope: Cluster
Version: v1

Параметры группы Huawei Cloud servers, которые будет использовать `CAPI` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

Пример:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

- **spec**
объект

Обязательный параметр

- **spec.additionalNetworks**

массив строк

Необязательный параметр.

Список ID сетей (подсетей) для дополнительных сетевых интерфейсов (NIC).

- **spec.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

Получить список всех доступных типов машин можно с помощью команды `cloud ECS ListFlavors`.

Для узлов всех видов, кроме master-узлов, желательно использовать тип виртуальных машин с локальным диском. Если в облаке поддерживаются локальные диски, они обычно быстрее и дешевле. Недостатком такого типа виртуальных машин является невозможность миграции узлов между гипервизорами.

Пример:

```
flavorName: m1.medium
```

- **spec.imageName**

строка

Необязательный параметр.

Имя образа виртуальной машины, который будет использоваться для инстанса.

Получить список всех доступных образов можно с помощью команды `cloud IMS ListImages`.

Список поддерживаемых Deckhouse ОС и их версий можно найти на странице Поддерживаемые версии Kubernetes и ОС (учитывайте используемую версию Deckhouse).

По умолчанию используется значение `HuaweiCloudDiscoveryData` или `instances.imageName`.

Пример:

```
imageName: ubuntu-18-04-cloud-amd64
```

- **spec.mainNetwork**

строка

Необязательный параметр.

ID основной сети для основного сетевого интерфейса (NIC) виртуальной машины.

По умолчанию используется значение из `HuaweiCloudDiscoveryData`.

- **spec.rootDiskSize**

целочисленный

Необязательный параметр.

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска.

Пример:

```
rootDiskSize: 20
```

- **spec.rootDiskType**

строка

Необязательный параметр.

Тип root-диска.

Параметр используется для указания типа тома, который будет использоваться для корневого диска.

По умолчанию используется значение `HuaweiCloudDiscoveryData`.

Пример:

```
rootDiskType: GPSSD
```

- **spec.securityGroups**

массив строк

Необязательный параметр.

Группы безопасности, которые будут применяться к сетевым портам виртуальных машин в дополнение к группам безопасности, установленным в конфигурации облачного провайдера.

Данные группы безопасности позволяют устанавливать правила брандмауэра для предоставленных инстансов.

Группа `SecurityGroups` может не поддерживаться поставщиком облачных услуг.

Пример:

```
securityGroups:
- security-group-1
- security-group-2
```

- **spec.serverGroupID**
строка

Необязательный параметр.

Группа серверов, в которую следует назначить машину.

Пример:

```
serverGroupID: server-group-1
```

- **spec.subnets**
Параметр устарел
массив строк

Необязательный параметр.

i Используйте поле `additionalNetworks` вместо этого параметра.

Пути к сетям, к которым будут подключаться вторичные сетевые карты `VirtualMachines`.

Пример:

```
subnets:
- BGP-network-VLAN-3894
- External-VLAN-3699
```

- **spec.vipAddress**
строка

Необязательный параметр.

Если заполнен, виртуальный IP-адрес будет назначен на все узлы инстанс-класса.

Допустимые значения: `Auto`, адрес IPv4.

Шаблон: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^Auto$`

Пример:

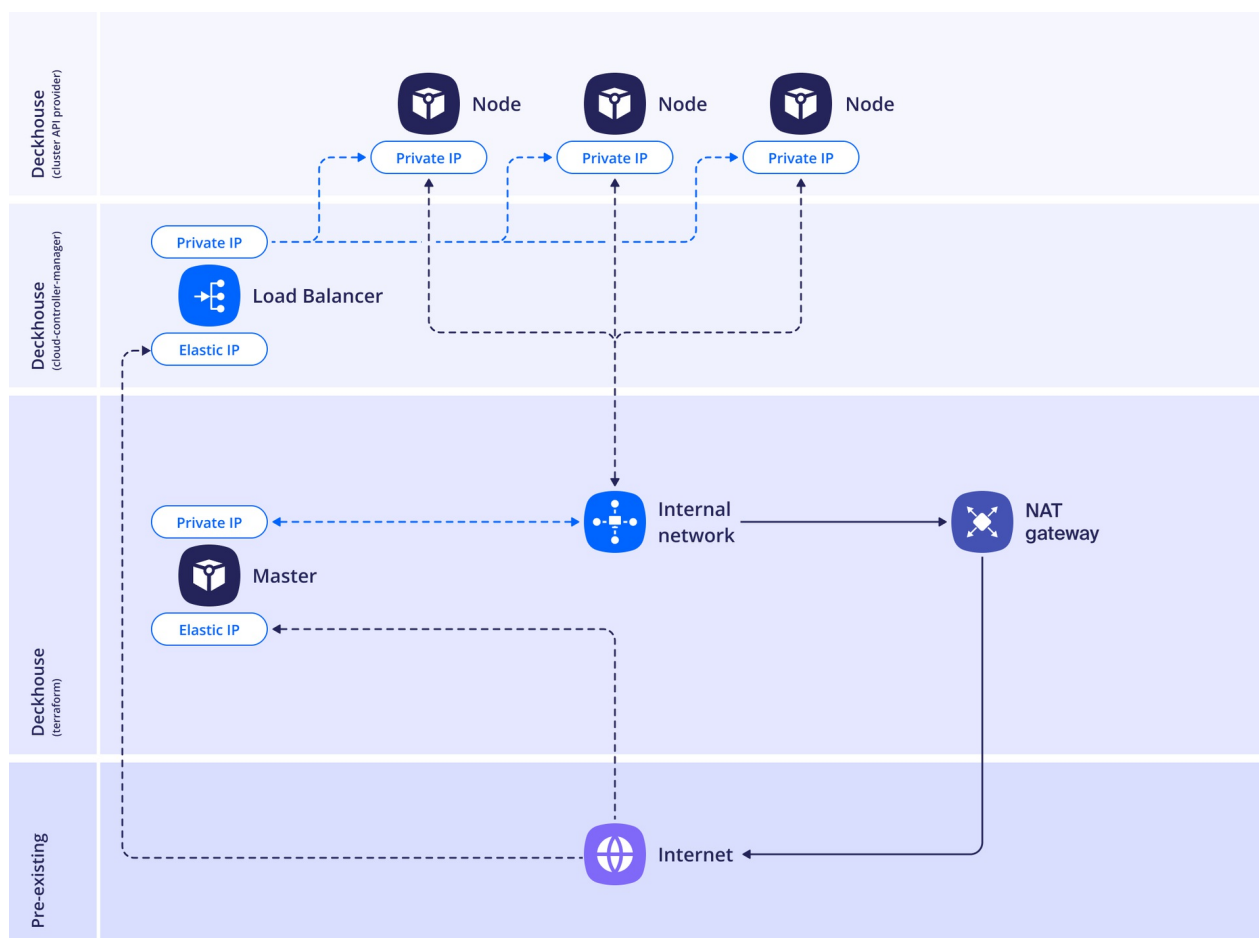
```
vipAddress: Auto
```

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

Схемы размещения

Standard

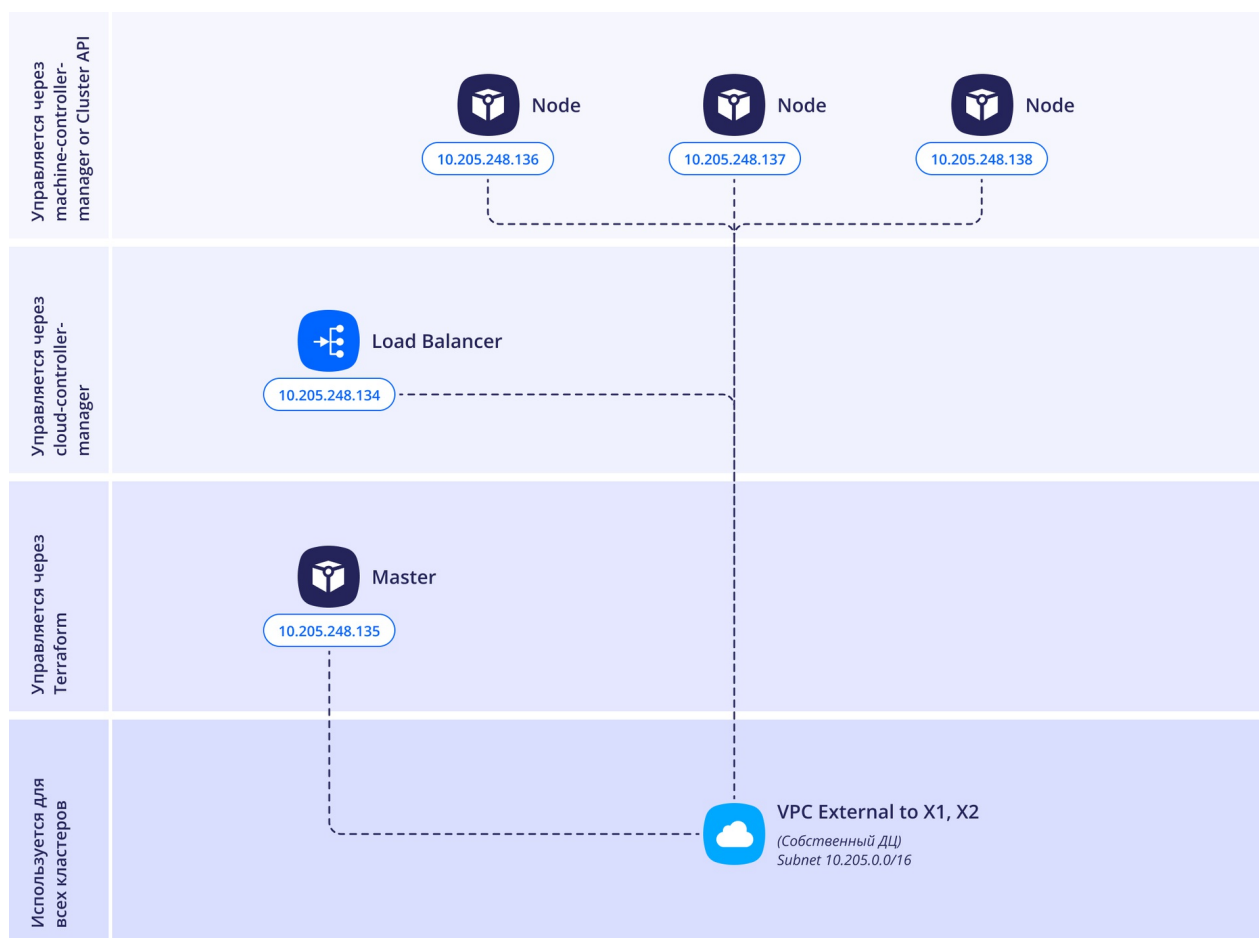
- Создается внутренняя сеть кластера со шлюзом к публичной сети.
- Elastic IP-адрес можно назначить master-узлу.
- Узлы, управляемые Cluster API, не имеют публичных IP-адресов.



Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSservers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering



Пример конфигурации схемы размещения:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSservers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4
```

Cloud provider — OpenStack

Модуль `cloud-provider-openstack` обеспечивает интеграцию Deckhouse Kubernetes Platform с облаками на базе OpenStack. Он предоставляет возможность модулю `node-manager` использовать ресурсы OpenStack при заказе узлов для `NodeGroup`.

Функции модуля `cloud-provider-openstack`:

- Управление ресурсами OpenStack через `cloud-controller-manager`:
 - актуализирует метаданные серверов OpenStack и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в OpenStack;
 - создаёт балансировщики нагрузки (Octavia) для Service с типом LoadBalancer.
- Заказ блочных дисков через CSI-драйвер Cinder (`cinder.csi.openstack.org`). Manila (filesystem) не поддерживается. CSI-драйвер Cinder поддерживает повторную аутентификацию в OpenStack с обновлением сервисного каталога, что повышает устойчивость длительно работающих подов с томами.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-openstack`.
- Заказ CloudEphemeral-узлов через Machine Controller Manager (MCM) или Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе `OpenStackInstanceClass`.
- Регистрация в модуле `node-manager`, чтобы `OpenStackInstanceClass` можно было указывать при описании `NodeGroup`.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`. Режим сети зависит от параметра `podNetworkMode`.



Модуль находится в процессе миграции управления CloudEphemeral-узлами с Machine Controller Manager (MCM) на Cluster API (CAPI). Существующие `NodeGroup` продолжают использовать MCM, а новые по умолчанию создаются с использованием CAPI. Порядок миграции существующих групп — в разделе «Как мигрировать группы узлов на Cluster API (CAPI)».

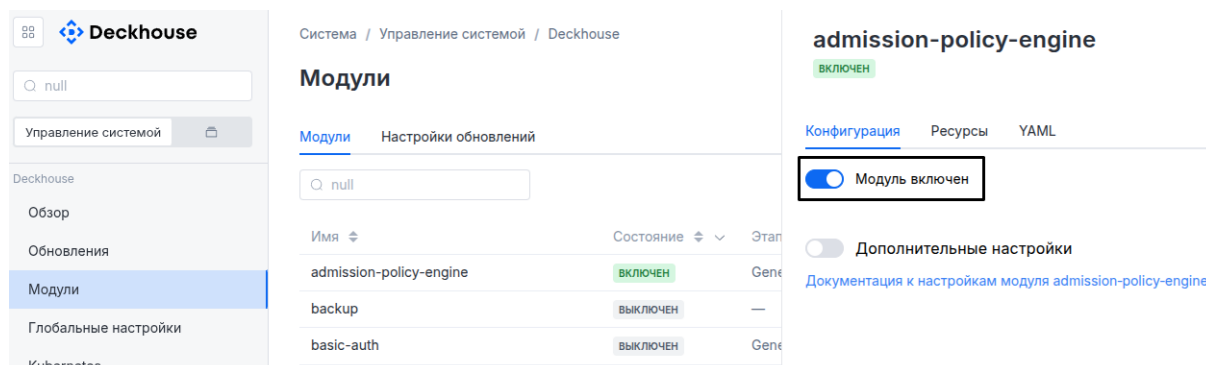
Настройки

Модуль автоматически включается для всех облачных кластеров, развернутых в OpenStack.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-openstack`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-openstack`:

```
d8 system module enable cloud-provider-openstack
```

- **С помощью ModuleConfig `cloud-provider-openstack`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-openstack` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-openstack`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  enabled: true
```

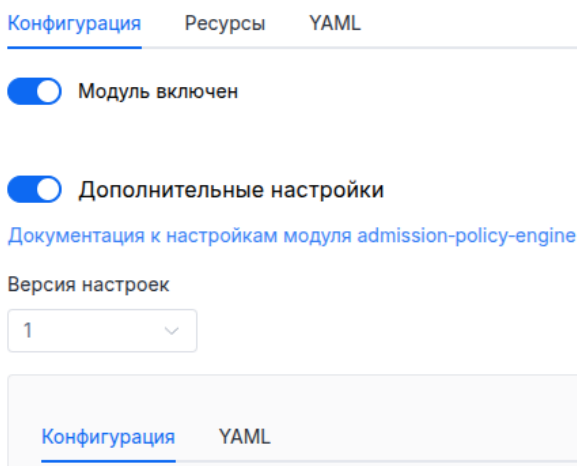
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-openstack`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `cloud-provider-openstack` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-openstack`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-openstack` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-openstack
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-openstack`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-openstack`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Количество и параметры процесса заказа машин в облаке настраиваются в custom resource `NodeGroup` модуля `node-manager`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference NodeGroup`). Инстанс-класс для cloud-провайдера OpenStack — это custom resource `OpenStackInstanceClass`, в котором указываются конкретные параметры самих машин.

i Кластер в OpenStack разворачивается в одном регионе (`provider.region` в `OpenStackClusterConfiguration`). Зоны других регионов использовать нельзя. Подробнее — в разделе «Как создать NodeGroup в зонах доступности?».

Настройки модуля устанавливаются автоматически на основании выбранной схемы размещения. В большинстве случаев нет необходимости в ручной конфигурации модуля.

Если вам необходимо настроить модуль, потому что, например, у вас bare-metal-кластер, для которого нужно включить возможность добавлять дополнительные инстансы из OpenStack, смотрите раздел Как настроить Hybrid-кластер в OpenStack.

i **Внимание.** При изменении настроек модуля **пересоздания существующих объектов Machines в кластере НЕ происходит** (новые объекты `Machine` будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров `NodeGroup` и `OpenStackInstanceClass`. См. подробнее в документации модуля `node-manager`. Для настройки аутентификации с помощью модуля `user-authn` необходимо в Crowd'е проекта создать новое `Generic` приложение.

Список необходимых сервисов OpenStack

Список сервисов OpenStack, необходимых для работы Deckhouse Kubernetes Platform в OpenStack:

Сервис	Версия API
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* Если нужно заказывать Load Balancer.

Чтобы интеграция и работа с платформой Deckhouse Kubernetes проходила корректно, присвойте ассоциированному пользователю OpenStack роль `member`. Это обеспечит разрешения на взаимодействие с указанными сервисами и управление ресурсами.

В случае использования публичных облаков, адреса и порты API можно узнать в официальной документации

сервиса:

- VK Cloud

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.additionalExternalNetworkNames**

массив строк

Имена дополнительных сетей, которые могут быть подключены к виртуальной машине и использованы в `cloud-controller-manager` для предоставления `ExternalIP` в `.status.addresses` в Node API объект.

Если в кластере есть инстансы, для которых будут использоваться External Networks, отличные от указанных в схеме размещения, их следует передавать в параметре `additionalExternalNetworkNames`.

Пример:

```
additionalExternalNetworkNames:
- some-bgp-network
```

- **settings.connection**

объект

Параметры подключения к API cloud-провайдера.

- **settings.connection.authURL**

строка

OpenStack Identity API URL.

- **settings.connection.caCert**

строка

CA x509 сертификат в PEM формате, использовавшийся для подписи (можно указать, если OpenStack API имеет самоподписанный сертификат).

- **settings.connection.domainName**

строка

Имя домена.

- **settings.connection.password**

строка

Пароль пользователя.

- **settings.connection.region**

строка

Регион OpenStack, где будет развернут кластер.

- **settings.connection.tenantID**

строка

ID проекта.

Не может использоваться вместе с `tenantName`.

- **settings.connection.tenantName**

строка

Имя проекта.

Не может использоваться вместе с `tenantID`.

- **settings.connection.username**

строка

Имя пользователя с полными правами на проект.

- **settings.csiDriver**

объект

Параметры настройки Cinder CSI Driver.

- **settings.csiDriver.fsGroupPolicy**

строка

Определяет, поддерживает ли том изменение владельца и прав доступа перед монтированием.

- `ReadWriteOnceWithFSType` — применять `fsGroup` только для томов `ReadWriteOnce` (поведение Kubernetes по умолчанию).
- `File` — всегда применять `fsGroup`. Используйте для `RWX`-томов с бэкендом Ceph.
- `None` — никогда не изменять права доступа.

По умолчанию: `ReadWriteOnceWithFSType`

Допустимые значения: `ReadWriteOnceWithFSType`, `File`, `None`

- **settings.externalNetworkNames**

массив строк

Имена сетей, подключенные к виртуальной машине и используемые `cloud-controller-manager` для предоставления ExternalIP в `.status.addresses` в Node API объект.

Пример:

```
externalNetworkNames:
- KUBE-3
- devops-internal
```

- **settings.ignoreVolumeMicroversion**

булевый

Настройка для обратной совместимости. Включите, если версия OpenStack в облаке меньше 3.34 и вы получаете ошибку «Version 3.34 is not supported by the API. Minimum is 3.0 and maximum is 3.x» при заказе PV. Это отключит изменения размеров томов «на лету», но вернет возможность заказа новых PV. (оригинальный PR <https://github.com/kubernetes/cloud-provider-openstack/pull/1986/>)

По умолчанию: `false`

- **settings.instances**

объект

Параметры инстансов, которые используются при создании виртуальных машин.

- **settings.instances.additionalNetworks**

массив строк

Список сетей, которые будут подключены к инстансу.

- **settings.instances.imageName**

строка

Имя образа.

- **settings.instances.mainNetwork**

строка

Путь до сети, которая будет подключена к виртуальной машине как основная сеть (шлюз по умолчанию).

- **settings.instances.securityGroups**

массив строк

Список securityGroups, которые нужно прикрепить к заказанным инстансам. Используется для задания firewall-правил по отношению к заказываемым инстансам.

- **settings.instances.sshKeyName**

строка

Имя OpenStack ресурса `keypair`, который будет использоваться при заказе инстансов.

- **settings.internalNetworkNames**

массив строк

Имена сетей, подключенные к виртуальной машине и используемые `cloud-controller-manager` для предоставления InternalIP в `.status.addresses` в Node API объект.

Пример:

```
internalNetworkNames:
- KUBE-3
- devops-internal
```

- **settings.loadBalancer**

объект

Параметры Load Balancer'a.

- **settings.loadBalancer.disabled**

булевый

Отключить поддержку LoadBalancer. Установите в `true`, если в облаке OpenStack отсутствует сервис Octavia (LBaaS).

По умолчанию: `false`

- **settings.loadBalancer.floatingNetworkID**

строка

ID external network, который будет использоваться для заказа Floating IP.

- **settings.loadBalancer.subnetID**

строка

ID подсети Neutron, в которой необходимо создать load balancer virtual IP.

- **settings.podNetworkMode**
строка

Способ организации трафика в той сети, которая используется для коммуникации между подами (обычно это internal-сеть, но бывают исключения):

- **DirectRouting** – между узлами работает прямая маршрутизация.
- **DirectRoutingWithPortSecurityEnabled** — между узлами работает прямая маршрутизация, но только если в OpenStack явно разрешить на портах диапазон адресов, используемых во внутренней сети:
 - **Внимание!** Убедитесь, что у `username` есть доступ на редактирование `AllowedAddressPairs` на портах, подключенных в сеть `internalNetworkName`. Обычно в OpenStack такого доступа нет, если сеть имеет флаг `shared`.
- **VXLAN** – между узлами **НЕ** работает прямая маршрутизация, необходимо использовать VXLAN.

Допустимые значения: `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`, `VXLAN`

- **settings.storageClass**
объект

Автоматическое создание StorageClass'ов, которые есть в OpenStack.

- **settings.storageClass.default**

Параметр устарел
строка

Имя StorageClass, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- произвольный StorageClass в кластере, имеющий default-аннотацию;
- первый (по алфавиту) StorageClass из тех, что создаются модулем (в порядке из OpenStack).

i **Параметр устарел.** Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: ceph-ssd
```

- **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) StorageClass'ов, которые не будут созданы в кластере.

Пример:

```
exclude:
- ".*-hdd"
- iscsi-fast
```

- **settings.storageClass.topologyEnabled**

булевый

Этот параметр управляет функционалом драйвера, который определяет необходимость учета ограничения топологии облака при заказе дисков. Это учитывается только при создании дисков, существующие PersistentVolume'ы не будут изменены.

i **Внимание.** Если установлено значение `false`, все новые PersistentVolume'ы создаются без учета ограничений топологии.

По умолчанию: `true`

- **settings.tags**
объект

Словарь тегов, которые будут на всех заказываемых инстансах.

- **settings.zones**
массив строк

Список зон по умолчанию для заказа инстансов. Может быть переопределен индивидуально для каждой NodeGroup.

Настройки провайдера

OpenStackClusterConfiguration

Version: `deckhouse.io/v1`

Описывает конфигурацию облачного кластера в OpenStack.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
- eu-3b
standard:
  internalNetworkDNSServers:
  - 8.8.8.8
  internalNetworkCIDR: 192.168.195.0/24
  internalNetworkSecurity: false
  externalNetworkName: external-network
provider:
  authURL: "<AUTH_URL>"
  domainName: "<DOMAIN_NAME>"
  tenantID: "<TENANT_ID>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  region: eu-3
masterNodeGroup:
  replicas: 1
  instanceClass:
    rootDiskSize: 50
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-3a: fast.eu-3a
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
    configDrive: false
    floatingIPPools:
    - public
    - shared
  additionalSecurityGroups:
  - sec_group_1
  - sec_group_2
zones:
- eu-3a
- eu-3b
```

- **apiVersion**
строка
Обязательный параметр
Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`
- **kind**
строка
Обязательный параметр
Допустимые значения: `OpenStackClusterConfiguration`
- **layout**
строка
Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

- **masterNodeGroup**

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `OpenStackInstanceClass`.

- **masterNodeGroup.instanceClass.additionalSecurityGroups**

массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего `OpenStackInstanceClass`. Default-группа также должна быть добавлена в этот список.

Используется для задания правил файрвола по отношению к заказываемым инстансам.

`SecurityGroups` могут не поддерживаться провайдером.

- **masterNodeGroup.instanceClass.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданным инстансам в дополнение к указанным в конфигурации облачного провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для etcd. Значение указывается в гигабайтах.

По умолчанию: `10`

Пример:

```
etcdDiskSizeGb: 10
```

- **masterNodeGroup.instanceClass.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

Чтобы узнать список всех доступных flavor'ов, выполните команду: `openstack flavor list`.

Для всех узлов, кроме master, рекомендуется выбирать flavor'ы с локальным диском. Такие диски, как правило, обеспечивают более высокую производительность и меньшую стоимость. Однако их использование имеет ограничение — невозможность миграции узлов между гипервизорами.

Пример создания flavor'a: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`.

- **masterNodeGroup.instanceClass.imageName**

строка

Обязательный параметр

Имя образа виртуальной машины, который будет использоваться для инстанса.

Получить список всех доступных образов можно командой: `openstack image list`.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

- **masterNodeGroup.instanceClass.rootDiskSize**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска. Подробнее...

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы количество было нечетным.

Допустимые значения: `1 <= X`

- **masterNodeGroup.serverGroup**
объект

Объединяет инстансы в группу. Инстансы в такой группе будут размещены на одном гипервизоре (affinity) или на разных гипервизорах (anti-affinity). Это позволяет повысить отказоустойчивость кластера.

- **masterNodeGroup.serverGroup.manuallyManaged**
объект

- **masterNodeGroup.serverGroup.manuallyManaged.id**
строка

Обязательный параметр

Идентификатор объекта ServerGroup.

- **masterNodeGroup.serverGroup.policy**
строка

Обязательный параметр

Политика распределения инстансов:

- `AntiAffinity` — размещает инстансы на разных гипервизорах.
- `ManuallyManaged` — позволяет указать существующую ServerGroup по идентификатору.

Допустимые значения: `AntiAffinity`, `ManuallyManaged`

- **masterNodeGroup.volumeTypeMap**
объект

Обязательный параметр

Словарь типов дисков для хранения данных etcd и конфигурационных файлов Kubernetes.

Формат элементов словаря: `<ЗОНА ДОСТУПНОСТИ>: <ТИП ДИСКА>` (см. пример).

Если указан параметр `rootDiskSize`, этот же тип диска будет использован в качестве загрузочного диска виртуальной машины.

Всегда рекомендуется использовать самые быстрые диски, предоставляемые провайдером.

Если значение, указанное в `replicas`, превышает количество элементов в словаре, master-узлы, чьи номера превышают длину словаря, получают значения, начиная с начала словаря. Если для словаря из примера указано `replicas: 5`, с типом диска `fast-eu-1a` будут master-0, master-2 и master-4, а с типом диска `fast-eu-1b` будут master-1 и master-3.

Полезные команды:

- `openstack availability zone list` — получить список зон доступности.
- `openstack volume type list` — получить список типов дисков.

Пример:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups**
массив объектов

Массив дополнительных NodeGroup для создания узлов типа CloudPermanent (например, для выделенных frontend-узлов или шлюзов).

Для узлов типа CloudEphemeral используйте отдельный ресурс NodeGroup и OpenStackInstanceClass.

- **nodeGroups.instanceClass**
объект

Частичное содержимое полей OpenStackInstanceClass.

- **nodeGroups.instanceClass.additionalNetworks**
массив строк

Список сетей, которые будут подключены к инстансу. Получить список доступных сетей можно командой: `openstack network list`.

Пример:

```
additionalNetworks:
- BGP-network-VLAN-3894
- External-VLAN-3699
```

■ **nodeGroups.instanceClass.additionalSecurityGroups**

массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего `OpenStackInstanceClass`. Default-группа также должна быть добавлена в этот список.

Используется для задания правил файрвола по отношению к заказываемым инстансам.

`SecurityGroups` могут не поддерживаться провайдером.

■ **nodeGroups.instanceClass.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданным инстансам в дополнение к указанным в конфигурации облачного провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

■ **nodeGroups.instanceClass.configDrive**

булевый

Необязательный параметр.

Флаг, указывающий, будет ли монтироваться на узел дополнительный диск, содержащий конфигурацию для узла.

Параметр необходимо устанавливать, если в сети, указанной в качестве `mainNetwork`, отключен DHCP.

По умолчанию: `false`

■ **nodeGroups.instanceClass.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

Чтобы узнать список всех доступных flavor'ов, выполните команду: `openstack flavor list`.

Для всех узлов, кроме master, рекомендуется выбирать flavor'ы с локальным диском. Такие диски, как правило, обеспечивают более высокую производительность и меньшую стоимость. Однако их использование имеет ограничение — невозможность миграции узлов между гипервизорами.

Пример создания flavor'a: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`.

■ **nodeGroups.instanceClass.floatingIPPools**

массив строк

Список сетей для заказа Floating IP для узлов.

■ **nodeGroups.instanceClass.imageName**

строка

Обязательный параметр

Имя образа виртуальной машины, который будет использоваться для инстанса.

Получить список всех доступных образов можно командой: `openstack image list`.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

■ **nodeGroups.instanceClass.mainNetwork**

строка

Обязательный параметр

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Получить список доступных сетей можно командой: `openstack network list`.

■ **nodeGroups.instanceClass.networksWithSecurityDisabled**

массив строк

Список сетей из параметров `mainNetwork` и `additionalNetworks`, в которых **запрещена** настройка `SecurityGroups` и `AllowedAddressPairs` на портах.

■ **nodeGroups.instanceClass.rootDiskSize**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска. Подробнее...

- **nodeGroups.name**

строка

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Количество узлов.

- **nodeGroups.volumeTypeMap**

объект

Словарь типов дисков для загрузочного диска.

Формат элементов словаря: `<ЗОНА ДОСТУПНОСТИ>: <ТИП ДИСКА>` (см. пример).

Если значение, указанное в `replicas`, превышает количество элементов в словаре, узлы, чьи номера превышают длину словаря, получают значения, начиная с начала словаря. Если для словаря из примера указано `replicas: 5`, с типом диска `fast-eu-1a` будут `worker-0`, `worker-2` и `worker-4`, а с типом диска `fast-eu-1b` будут `worker-1` и `worker-3`.

i **Внимание.** Ограничение по зонам доступности работает только при использовании параметра `zones` (`nodeGroups.zones`). Если параметр `zones` не задан, будут использованы все зоны доступности.

Master-узлы будут создаваться в зонах в алфавитном порядке, указанных в этом параметре. Исходя из примера ниже, `master-0` будет создан в зоне `eu-1a`, `master-1` будет создан в зоне `eu-1b`, а `master-2` будет создан в зоне `eu-1a`.

Полезные команды:

- `openstack availability zone list` — получить список зон доступности.
- `openstack volume type list` — получить список типов дисков.

Пример:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups.zones**

массив строк

Необязательный параметр.

Список зон, в которых допустимо создавать узлы.

Зоны должны относиться к региону `provider.region`. Если задан параметр `zones`, можно указать только зоны из этого параметра.

- **provider**

объект

Обязательный параметр

Параметры подключения к API OpenStack.

Совпадают с параметрами, передаваемыми в поле `connection` в модуле `cloud-provider-openstack`.

- **provider.authURL**

строка

OpenStack Identity API URL.

- **provider.caCert**

строка

CA x509 сертификат, использовавшийся для подписи (можно указать, если OpenStack API имеет самоподписанный сертификат).

Пример:

```
caCert: |
  -----BEGIN CERTIFICATE-----
  MIIFyDCCBLCgAwIBAgIQBwDIWH1asdaKNaALUa4NUzANBgqhkiG9w0BAQsFADBc
  ...
  -----END CERTIFICATE-----
```

- **provider.domainName**

строка

Имя домена.

Переменная `OS_USER_DOMAIN_NAME` из файла `openrc`.

- **provider.password**

строка

Пароль пользователя.

- **provider.region**

строка

Регион OpenStack, в котором будет развернут кластер.

Кластер работает только в одном регионе. Master-узлы, а также узлы типов `CloudPermanent` и `CloudEphemeral` можно создавать только в зонах этого региона.

- **provider.tenantID**

строка

ID проекта.

Не может использоваться вместе с `tenantName`.

- **provider.tenantName**

строка

Имя проекта.

Не может использоваться вместе с `tenantID`.

- **provider.username**

строка

Имя пользователя с полными правами на проект.

- **simple**
объект

Настройки для схемы размещения `Simple`.

- **simple.externalNetworkDHCP**
булевый

Флаг, который указывает, включен ли DHCP в сети, указанной в качестве внешней.

По умолчанию: `true`

- **simple.externalNetworkName**
строка

Обязательный параметр

Имя сети для внешнего взаимодействия. Получить список доступных сетей можно командой: `openstack network list`.

- **simple.podNetworkMode**
строка

Определяет способ организации трафика в той сети, которая используется для коммуникации между подами (обычно это внутренняя сеть, но бывают исключения):

- `DirectRouting` – между узлами работает прямая маршрутизация, в этом режиме отключены `SecurityGroups`.
- `VXLAN` – между узлами **не работает** прямая маршрутизация, необходимо использовать `VXLAN`, в этом режиме отключены `SecurityGroups`.

i **Внимание.** После изменения этого параметра необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

i **Внимание.** После переключения с/на `VXLAN` требуется перезагрузка всех узлов кластера.

По умолчанию: `VXLAN`

Допустимые значения: `VXLAN`, `DirectRouting`

- **simpleWithInternalNetwork**
объект

Настройки для схемы размещения `SimpleWithInternalNetwork`.

- **simpleWithInternalNetwork.externalNetworkDHCP**
булевый

Флаг, который указывает, включен ли DHCP в сети, указанной в качестве внешней.

По умолчанию: `true`

- **simpleWithInternalNetwork.externalNetworkName**
строка

Имя сети для внешнего взаимодействия. Получить список доступных сетей можно командой: `openstack network list`.

- **simpleWithInternalNetwork.internalSubnetName**
строка

Обязательный параметр

Имя подсети, в которой будут работать узлы кластера.

- **simpleWithInternalNetwork.masterWithExternalFloatingIP**
булевый

Флаг, который указывает, создавать ли Floating IP на master-узлах.

По умолчанию: `true`

- **simpleWithInternalNetwork.podNetworkMode**
строка

Определяет способ организации трафика в той сети, которая используется для коммуникации между подами (обычно это внутренняя сеть, но бывают исключения):

- `DirectRouting` — между узлами работает прямая маршрутизация, в этом режиме отключены `SecurityGroups`.
- `DirectRoutingWithPortSecurityEnabled` — между узлами работает прямая маршрутизация, но только если в OpenStack явно разрешить на портах диапазон адресов, используемых во внутренней сети.
 - **Внимание.** Убедитесь, что у `username` есть доступ на редактирование `AllowedAddressPairs` на портах, подключенных в сеть `internalNetworkName`. Обычно в OpenStack такого доступа нет, если сеть имеет флаг `shared`.

- **VXLAN** — между узлами **не работает** прямая маршрутизация, необходимо использовать VXLAN, в этом режиме отключены SecurityGroups.

i **Внимание.** После изменения этого параметра необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу.

i **Внимание.** После переключения с/на VXLAN требуется перезагрузка всех узлов кластера.

По умолчанию: `DirectRoutingWithPortSecurityEnabled`

Допустимые значения: `VXLAN`, `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`

- **sshAllowList**

массив строк

Список CIDR, разрешенных для подключения к узлам по SSH.

По умолчанию `0.0.0.0/0`.

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **standard**

объект

Настройки для схемы размещения `Standard`.

- **standard.bastion**

объект

Спецификация для описания bastion-узла.

- **standard.bastion.instanceClass**

объект

Частичное содержимое полей `OpenStackInstanceClass`.

- **standard.bastion.instanceClass.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданному инстансу в дополнение к указанным в конфигурации облачного провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

- **standard.bastion.instanceClass.flavorName**

строка

Обязательный параметр

Тип виртуальной машины.

Чтобы узнать список всех доступных flavor'ов, выполните команду: `openstack flavor list`.

- **standard.bastion.instanceClass.imageName**

строка

Обязательный параметр

Имя образа виртуальной машины, который будет использоваться для инстанса.

Чтобы узнать список всех доступных образов, выполните команду: `openstack image list`.

- **standard.bastion.instanceClass.rootDiskSize**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска. Подробнее...

По умолчанию: `50`

- **standard.bastion.volumeType**

строка

Тип корневого диска.

- **standard.bastion.zone**

строка

Зона для создания инстанса для bastion-узла.

- **standard.externalNetworkName**

строка

Обязательный параметр

Имя сети для внешнего взаимодействия. Получить список доступных сетей можно командой: `openstack network list`.

- **standard.internalNetworkCIDR**

строка

Обязательный параметр

Адресация для внутренней сети узлов кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standard.internalNetworkDNSServers**

массив строк

Обязательный параметр

Список адресов рекурсивных DNS внутренней сети.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standard.internalNetworkSecurity**

булевый

Флаг, который определяет необходимость настройки SecurityGroups и AllowedAddressPairs на портах внутренней сети.

По умолчанию: `true`

- **standardWithNoRouter**

объект

Настройки для схемы размещения `StandardWithNoRouter`.

- **standardWithNoRouter.externalNetworkDHCP**

булевый

Флаг, который указывает, включен ли DHCP в сети, указанной в качестве внешней.

По умолчанию: `true`

- **standardWithNoRouter.externalNetworkName**

строка

Обязательный параметр

Имя сети для внешнего взаимодействия. Получить список доступных сетей можно командой: `openstack network list`.

- **standardWithNoRouter.internalNetworkCIDR**

строка

Обязательный параметр

Адресация для внутренней сети узлов кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standardWithNoRouter.internalNetworkSecurity**

булевый

Флаг, который определяет необходимость настройки SecurityGroups и AllowedAddressPairs на портах внутренней сети.

По умолчанию: `true`

- **tags**

объект

Необязательный параметр.

Список тегов, которые будут созданы на всех ресурсах, имеющих такую возможность.

Если поменять теги в рабочем кластере, после применения изменений необходимо пересоздать все машины.

- **zones**

массив строк

Необязательный параметр.

Глобальное ограничение набора зон доступности, с которыми работает облачный провайдер.

Указывайте только зоны региона `provider.region`. Кластер разворачивается в одном регионе OpenStack — зоны других регионов использовать нельзя.

Получить список зон региона можно командой: `openstack availability zone list --compute`.

Подготовка окружения



Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Чтобы Deckhouse мог управлять ресурсами в облаке OpenStack, ему необходимо подключиться к OpenStack API. Перечень API-сервисов OpenStack, доступ к которым необходим для развертывания, доступен в разделе настройки.

Доступы пользователя, необходимые для подключения к OpenStack API, находятся в `openrc`-файле (OpenStack RC file).

Информация о получении `openrc`-файла с помощью стандартного веб-интерфейса OpenStack и о способах его использования доступна в документации OpenStack.

Если вы используете OpenStack API облачного провайдера, интерфейс получения `openrc`-файла может быть другим.

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

OpenStackInstanceClass

Scope: Cluster

- `v1`
- `v1alpha1`

Параметры группы OpenStack servers, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalNetworks**
массив строк

Список сетей, которые будут подключены к инстансу.

По умолчанию используется значение из `OpenStackCloudDiscoveryData`.

Пример:

```
additionalNetworks:
- BGP-network-VLAN-3894
- External-VLAN-3699
```

- **spec.additionalSecurityGroups**

массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего OpenStackInstanceClass. Default-группа также должна быть добавлена в этот список.

Используется для задания правил firewall по отношению к заказываемым инстансам.

SecurityGroups могут не поддерживаться провайдером.

Пример:

```
additionalSecurityGroups:
- security-group-1
- security-group-2
```

◦ **spec.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданным инстансам в дополнение к указанным в конфигурации cloud-провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

◦ **spec.capacity**

Параметр устарел

объект

Устарело: параметр больше не используется. Для переданного типа инстанса Deckhouse использует параметры, полученные из облака.

Емкость инстанса для *нестандартных* типов инстансов (параметр `flavorName`).

Используется cluster-autoscaler'ом при планировании, только когда в NodeGroup'е еще нет узлов (при `minPerZone` равно 0). Если в NodeGroup уже есть узлы, cluster-autoscaler использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов инстансов. Параметры *стандартных* типов инстансов (`m1.medium`, `m1.large` и т. п.) заложены в Deckhouse.

▪ **spec.capacity.cpu**

строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

▪ **spec.capacity.memory**

строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

◦ **spec.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

Получить список всех доступных flavor'ов можно с помощью команды: `openstack flavor list`.

Для всех не master-узлов желательно использовать flavor'ы с локальным диском. Если в облаке поддерживаются локальные диски, они обычно быстрее и дешевле. Недостатком использования таких flavor'ов является невозможность миграции узлов между гипервизорами.

Пример создания flavor'a: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

Пример:

```
flavorName: m1.medium
```

- **spec.imageName**

строка

Необязательный параметр.

Имя образа виртуальной машины, который будет использоваться для инстанса.

Получить список всех доступных образов можно с помощью команды: `openstack image list`.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте используемую версию Deckhouse).

По умолчанию используется значение либо из `OpenStackCloudDiscoveryData`, либо из настроек `instances.imageName`.

Пример:

```
imageName: ubuntu-18-04-cloud-amd64
```

- **spec.mainNetwork**

строка

Необязательный параметр.

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

По умолчанию используется значение из `OpenStackCloudDiscoveryData`.

Пример:

```
mainNetwork: k8s-msk-178
```

- **spec.rootDiskSize**

целочисленный

Необязательный параметр.

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска. Подробнее...

Пример:

```
rootDiskSize: 20
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

Параметры группы OpenStack servers, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**

объект

Обязательный параметр

- **spec.additionalNetworks**

массив строк

Список сетей, которые будут подключены к инстансу.

По умолчанию используется значение из `OpenStackCloudDiscoveryData`.

Пример:

```
additionalNetworks:
- BGP-network-VLAN-3894
- External-VLAN-3699
```

- **spec.additionalSecurityGroups**

массив строк

Дополнительный список security groups, которые будут добавлены на заказанные инстансы соответствующего `OpenStackInstanceClass`. Default-группа также должна быть добавлена в этот список.

Используется для задания правил firewall по отношению к заказываемым инстансам.

`SecurityGroups` могут не поддерживаться провайдером.

Пример:

```
additionalSecurityGroups:
```

- security-group-1
- security-group-2

◦ **spec.additionalTags**

объект

Дополнительные теги, которые будут присвоены созданным экземплярам в дополнение к указанным в конфигурации cloud-провайдера.

Пример:

```
additionalTags:
  project: cms-production
  severity: critical
```

◦ **spec.capacity**

Параметр устарел
объект

Устарело: параметр больше не используется. Для переданного типа экземпляра Deckhouse использует параметры, полученные из облака.

Емкость экземпляра для *нестандартных* типов экземпляров (параметр `flavorName`).

Используется cluster-autoscaler'ом при планировании, только когда в NodeGroup'е еще нет узлов (при minPerZone равно 0). Если в NodeGroup уже есть узлы, cluster-autoscaler использует при планировании фактические данные (CPU, memory) о мощности узла и не использует данные параметра `capacity`.

Параметр необходимо указывать только для *нестандартных* типов экземпляров. Параметры *стандартных* типов экземпляров (`m1.medium`, `m1.large` и т. п.) заложены в Deckhouse.

■ **spec.capacity.cpu**

строка или число

Обязательный параметр

Количество vCPU узла.

Шаблон: `^[0-9]+m?$`

Пример:

```
cpu: 1000m
```

■ **spec.capacity.memory**

строка или число

Обязательный параметр

Количество памяти узла.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
memory: 1500Mi
```

◦ **spec.flavorName**

строка

Обязательный параметр

Тип заказываемых виртуальных машин.

Получить список всех доступных flavor'ов можно с помощью команды: `openstack flavor list`.

Для всех не master-узлов желательно использовать flavor'ы с локальным диском. Если в облаке поддерживаются локальные диски, они обычно быстрее и дешевле. Недостатком использования таких flavor'ов является невозможность миграции узлов между гипервизорами.

Пример создания flavor'a: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

Пример:

```
flavorName: m1.medium
```

◦ **spec.imageName**

строка

Необязательный параметр.

Имя образа виртуальной машины, который будет использоваться для экземпляра.

Получить список всех доступных образов можно с помощью команды: `openstack image list`.

Список поддерживаемых Deckhouse ОС и их версий можно найти в документации (учитывайте

используемую версию Deckhouse).

По умолчанию используется значение либо из `OpenStackCloudDiscoveryData`, либо из настроек `instances.imageName`.

Пример:

```
imageName: ubuntu-18-04-cloud-amd64
```

- **spec.mainNetwork**

строка

Необязательный параметр.

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

По умолчанию используется значение из `OpenStackCloudDiscoveryData`.

Пример:

```
mainNetwork: k8s-msk-178
```

- **spec.rootDiskSize**

целочисленный

Необязательный параметр.

Размер root-диска. Значение указывается в гигабайтах.

Параметр также влияет на тип диска. Подробнее...

Пример:

```
rootDiskSize: 20
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

FAQ

Как настроить LoadBalancer?



Для корректного определения клиентского IP-адреса необходимо использовать LoadBalancer с поддержкой Proxy Protocol.

Рекомендуется ограничивать список узлов, добавляемых в пул балансировщика, с помощью аннотации `loadbalancer.openstack.org/node-selector`.

Без ограничения по `node-selector` cloud-controller-manager может использовать в качестве таргетов балансировщика все подходящие узлы кластера. В результате добавление или удаление узлов, не связанных с обслуживаемой балансировщиком нагрузкой, может приводить к обновлению состава пула балансировщика. В крупных или часто изменяющихся кластерах такие обновления могут происходить регулярно, а в некоторых конфигурациях сопровождаться кратковременными нарушениями существующих соединений.

С помощью `loadbalancer.openstack.org/node-selector` рекомендуется выбирать только те узлы, которые должны использоваться в качестве таргетов данного LoadBalancer.

Пример IngressNginxController

В примере поды Ingress-контроллера размещаются на frontend-узлах, а аннотация `loadbalancer.openstack.org/node-selector` ограничивает пул балансировщика этими же узлами:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend

```

Как настроить политики безопасности на узлах кластера?

Вариантов, зачем может понадобиться ограничить или, наоборот, расширить входящий или исходящий трафик на виртуальных машинах кластера, может быть множество. Например:

- Разрешить подключение к узлам кластера с виртуальных машин из другой подсети.
- Разрешить подключение к портам статического узла для работы приложения.
- Ограничить доступ к внешним ресурсам или другим ВМ в облаке по требованию службы безопасности.

Для всего этого следует применять дополнительные группы безопасности (security groups). Можно использовать только группы безопасности, предварительно созданные в облаке.

Установка дополнительных групп безопасности (security groups) на статических и master-узлах

Данный параметр можно задать либо при создании кластера, либо в уже существующем кластере. В обоих случаях дополнительные группы безопасности указываются в `OpenStackClusterConfiguration`:

- для master-узлов — в секции `masterNodeGroup` в поле `additionalSecurityGroups`;
- для статических узлов — в секции `nodeGroups` в конфигурации, описывающей желаемую `nodeGroup`, а также в поле `additionalSecurityGroups`.

Поле `additionalSecurityGroups` представляет собой массив строк с именами групп безопасности.

Установка дополнительных групп безопасности (security groups) на ephemeral-узлах

Необходимо прописать параметр `additionalSecurityGroups` для всех `OpenStackInstanceClass` в кластере, которым нужны дополнительные группы безопасности. Подробнее — параметры модуля `cloud-provider-openstack`.

Как создать NodeGroup в зонах доступности?

Кластер OpenStack разворачивается в одном регионе, который задается параметром `provider.region` ресурса `OpenStackClusterConfiguration`. Создавать узлы можно только в зонах доступности этого региона. Использование зон из других регионов не поддерживается.

Чтобы получить список зон доступности региона, выполните команду:

```
openstack --os-region-name <REGION> availability zone list --compute
```

Группы узлов типа `CloudPermanent` задаются в секции `nodeGroups` ресурса `OpenStackClusterConfiguration`. Для ограничения зон доступности используйте параметр `nodeGroups[].zones`.

Узлы типа `CloudEphemeral` создаются отдельным ресурсом `NodeGroup` с `nodeType: CloudEphemeral`. Зоны доступности для них задаются в параметре `spec.cloudInstances.zones`. Если в `OpenStackClusterConfiguration` указан параметр `zones`, необходимые зоны также должны быть добавлены в него.



Если при создании `NodeGroup` адмиссия вебхук возвращает ошибку `unknown zone`, убедитесь, что указанная зона относится к региону `provider.region` и, при использовании параметра `zones`, включена в список зон `OpenStackClusterConfiguration`.

Узлы из другого региона можно добавить в кластер только вручную как `Static`-узлы. Модуль `cloud-provider-openstack` не создает такие узлы.



После изменения `OpenStackClusterConfiguration` выполните команду `dhctl converge`. Подробнее — в разделе «Добавление и управление облачными узлами».

Пример NodeGroup для узлов типа CloudEphemeral:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: workers
  minPerZone: 1
  maxPerZone: 1
  zones:
    - eu-3a
    - eu-3b
```

Как поднять гибридный кластер?

Гибридный кластер представляет собой кластер, в котором могут быть как узлы bare metal, так и узлы OpenStack. Для создания такого кластера необходимо наличие L2-сети между всеми узлами кластера.

i В Deckhouse Kubernetes Platform есть возможность задавать префикс для имени CloudEphemeral-узлов, добавляемых в гибридный кластер с master-узлами типа Static. Для этого используйте параметр `instancePrefix` модуля `node-manager`. Префикс, указанный в параметре, будет добавляться к имени всех добавляемых в кластер узлов типа CloudEphemeral. Задать префикс для определенной NodeGroup нельзя.

Чтобы поднять гибридный кластер, выполните следующие шаги:

1. Удалите flannel из kube-system: `d8 k -n kube-system delete ds flannel-ds`.
2. Включите и настройте модуль.
3. Создайте один или несколько custom resource OpenStackInstanceClass.
4. Создайте один или несколько custom resource NodeManager для управления количеством и процессом заказа машин в облаке.

i **Важно!** Cloud-controller-manager синхронизирует состояние между OpenStack и Kubernetes, удаляя из Kubernetes те узлы, которых нет в OpenStack. В гибридном кластере такое поведение не всегда соответствует потребности, поэтому, если узел Kubernetes запущен не с параметром `--cloud-provider=external`, он автоматически игнорируется (Deckhouse прописывает `static://` на узлы в `.spec.providerID`, а cloud-controller-manager такие узлы игнорирует).

Подключение storage в гибридном кластере

Если вам требуются PersistentVolumes на узлах, подключаемых к кластеру из OpenStack, необходимо создать StorageClass с нужным OpenStack volume type. Получить список типов можно с помощью команды `openstack volume type list`.

Например, для volume type `ceph-ssd`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ceph-ssd
provisioner: csi-cinderplugin # Обязательно должно быть так.
parameters:
  type: ceph-ssd
volumeBindingMode: WaitForFirstConsumer
```

Как загрузить образ в OpenStack?

1. Скачайте последний стабильный образ Ubuntu 18.04:

```
curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
```

2. Подготовьте OpenStack openrc-файл, который содержит credentials для обращения к API OpenStack.

i Интерфейс получения openrc-файла может отличаться в зависимости от провайдера OpenStack. Если провайдер предоставляет стандартный интерфейс для OpenStack, скачать openrc-файл можно по инструкции.

3. Либо установите OpenStack-клиента по инструкции.

Также можно запустить контейнер, смонтировать в него openrc-файл и скачанный локально образ Ubuntu:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Инициализируйте переменные окружения из openrc-файла:

```
source /openrc
```

5. Получите список доступных типов дисков:

```
/ # openstack volume type list
+-----+-----+-----+
| ID | Name | Is Public |
+-----+-----+-----+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | kol-high-iops | True |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True |
| 74101409-a462-4f03-872a-7de727a178b8 | kol-ssd | True |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd | True |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd | True |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1 | True |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1 | True |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph | True |
+-----+-----+-----+
```

6. Создайте образ и передайте в него в качестве свойств тип диска, который будет использоваться (если OpenStack не поддерживает локальные диски или если эти диски не подходят для работы):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Проверьте, что образ успешно создан:

```
/ # openstack image show ubuntu-18-04-cloud-amd64
+-----+-----+
| Field | Value |
+-----+-----+
| checksum | 3443a1fd810f4af9593d56e0e144d07d |
| container_format | bare |
| created_at | 2020-01-10T07:23:48Z |
| disk_format | qcow2 |
| file | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file |
| id | 01998f40-57cc-4ce3-9642-c8654a6d14fc |
| min_disk | 0 |
| min_ram | 0 |
| name | ubuntu-18-04-cloud-amd64 |
| owner | bb5f506e3ece54e21b2acf1bf9db4f62c |
| properties | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url': u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', u'metadata': {}}]' |
| protected | False |
| schema | /v2/schemas/image |
| size | 343277568 |
| status | active |
| tags | |
| updated_at | 2020-05-01T17:18:34Z |
| virtual_size | None |
| visibility | private |
+-----+-----+
```

Как проверить, поддерживает ли провайдер группы безопасности (security groups)?

Достаточно выполнить команду `openstack security group list`. Если в ответ вы не получите ошибок, это значит, что группы безопасности поддерживаются.

Как настроить работу онлайн-изменения размера дисков?

OpenStack API успешно рапортует об изменении размера диска, но Cinder никак не оповещает Nova о том, что диск изменился, поэтому диск внутри гостевой ОС остается старого размера.

Для устранения проблемы необходимо прописать в `cinder.conf` параметры доступа к Nova API. Например, так:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Источник...

Как использовать `rootDiskSize` и когда он предпочтителен?

Диски в OpenStack

Диск узла может быть локальным или сетевым. В терминологии OpenStack локальный диск — это ephemeral disk, а сетевой — persistent disk (cinder storage). Узлы с локальными дисками не могут мигрировать между гипервизорами.

- Для master-узла предпочтительнее сетевой диск, чтобы узел мог мигрировать между гипервизорами.
- Для ephemeral-узла предпочтительнее локальный диск, чтобы сэкономить на стоимости. Не все cloud-провайдеры поддерживают использование локальных дисков. Если локальные диски не поддерживаются, для ephemeral-узлов придется использовать сетевые диски.

Локальный диск (ephemeral)	Сетевой диск (persistent)
Дешевле	Дороже
Подходит для ephemeral-узлов	Подходит для master-узлов

Параметр `rootDiskSize`

В `OpenStackInstanceClass` есть параметр `rootDiskSize`, и в OpenStack flavor есть параметр размера диска.

Какой диск закажется в зависимости от комбинации параметров, указано в таблице:

	flavor disk size = 0	flavor disk size > 0
rootDiskSize не указан	❑ Необходимо задать размер. Без указания размера будет ошибка создания VM.	Локальный диск с размером из flavor
rootDiskSize указан	Сетевой диск размером <code>rootDiskSize</code>	❑ Сетевой (<code>rootDiskSize</code>) и локальный (из flavor). Избегайте использования этого варианта, так как cloud-провайдер будет взимать плату за оба диска.

i Обратите внимание, что при создании узлов с типом `CloudEphemeral` в облаке Selectel, для создания узла в зоне отличной от зоны А, необходимо заранее создать flavor с диском необходимого размера. Параметр `rootDiskSize` в этом случае указывать не нужно.

Рекомендация для master-узлов и бастiona — сетевой диск

- Используйте flavor с нулевым размером диска.
- Задайте `rootDiskSize` в `OpenStackInstanceClass`.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, тип диска будет взят из `volumeTypeMap`.

Рекомендация для ephemeral-узлов — локальный диск

- Используйте flavor с заданным размером диска.
- Не используйте параметр `rootDiskSize` в `OpenStackInstanceClass`.
- Проконтролируйте тип диска. Тип диска будет взят из образа ОС, если он задан. Если нет, будет использоваться тип диска по умолчанию cloud-провайдера.

Как проверить объем диска в flavor?

```
# openstack flavor show m1.medium-50g -c disk
+-----+-----+
| Field | Value |
+-----+-----+
| disk  | 50    |
+-----+-----+
```

Как переопределить тип диска по умолчанию cloud-провайдера?

Если у cloud-провайдера доступно несколько типов, вы можете указать тип диска по умолчанию для образа. Для этого необходимо указать название типа диска в метаданных образа. Тогда все VM, создаваемые из этого образа, будут использовать указанный тип сетевого диска.

Также вы можете создать новый образ OpenStack следующим образом.

Пример:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Офлайн-изменение размера диска

Некоторые облачные провайдеры (например, VK Cloud) могут не поддерживать онлайн-изменение дисков. Если при изменении размера диска вы получаете следующую ошибку, необходимо уменьшить количество реплик StatefulSet до 0, подождать изменения размера дисков и вернуть обратно количество реплик, которое было до начала операции.

```
Warning VolumeResizeFailed 5s (x11 over 41s) external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/5555555555555555/volumes/bb5a275b-3f30-4916-9480-9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Что делать, если переключение на заказ узлов в менее приоритетных группах занимает много времени?

Если переключение на заказ узлов в менее приоритетных группах занимает много времени, воспользуйтесь инструкцией.

Примеры

Ниже представлены два простых примера конфигурации cloud-провайдера OpenStack.

Пример 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Пример 2

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
      - public
    internalNetworkNames:
      - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
        - default
        - allow-ssh-and-icmp
    zones:
      - zone-a
      - zone-b
    tags:
      project: cms
      owner: default

```

Схемы размещения

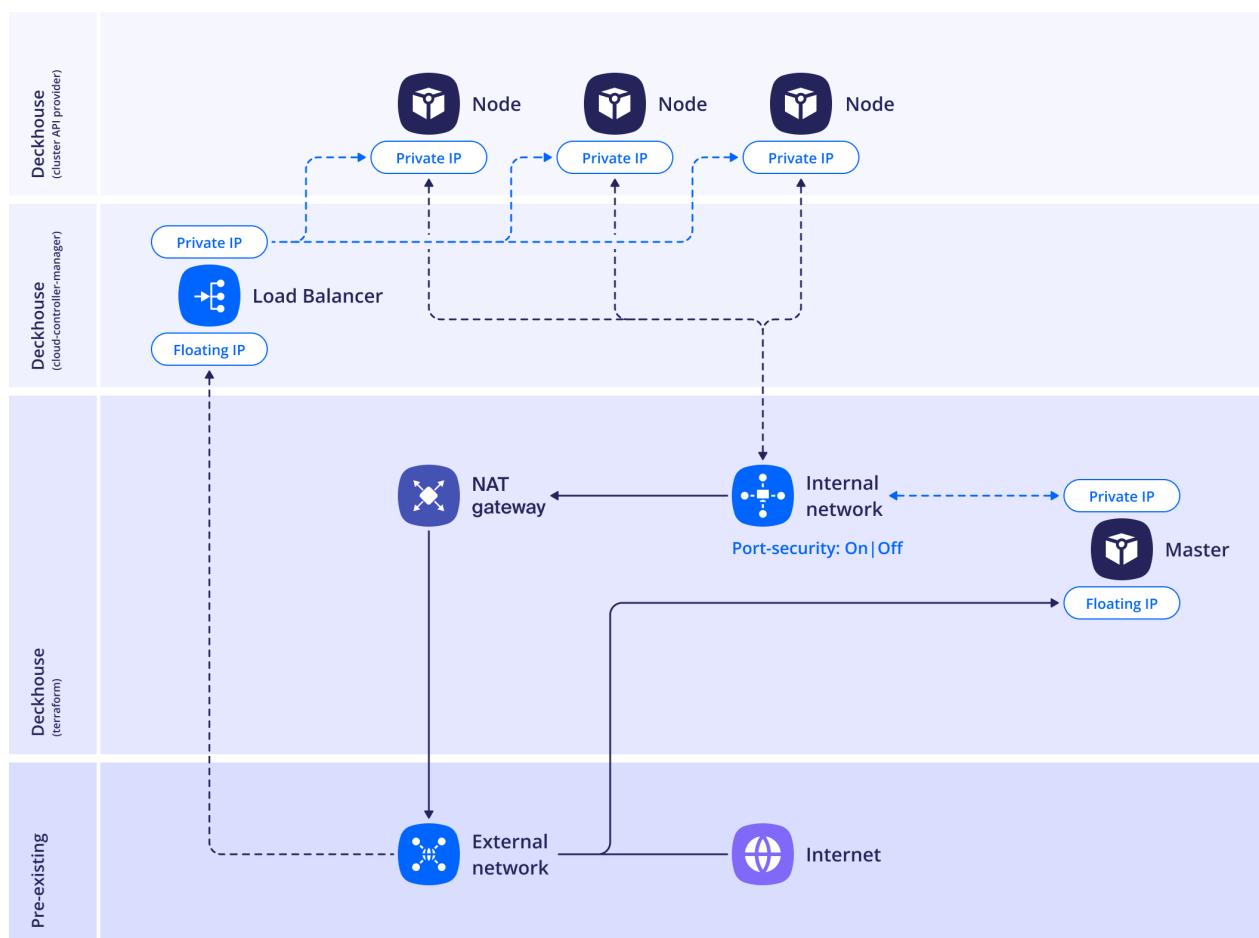
Поддерживаются четыре схемы размещения. Ниже подробнее о каждой из них.

Standard

Создается внутренняя сеть кластера со шлюзом в публичную сеть, узлы не имеют публичных IP-адресов. Для master-узла заказывается плавающий IP-адрес.

i

Внимание. Если провайдер не поддерживает SecurityGroups, все приложения, запущенные на узлах с Floating IP, будут доступны по белому IP-адресу. Например, kube-apiserver на master-узлах будет доступен на порту 6443. Чтобы избежать этого, рекомендуется использовать схему размещения SimpleWithInternalNetwork, либо Standard с bastion-узлом.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  internalNetworkDNSServers:                # Обязательный параметр.
    - 8.8.8.8
    - 4.2.2.2
  internalNetworkSecurity: true|false        # Необязательный параметр, по умолчанию true.
  externalNetworkName: shared               # Обязательный параметр.
  bastion:
    zone: ru2-b                             # Необязательный параметр.
    volumeType: fast-ru-2b                  # Необязательный параметр.
    instanceClass:
      flavorName: m1.large                   # Обязательный параметр.
      imageName: ubuntu-20-04-cloud-amd64   # Обязательный параметр.
      rootDiskSize: 50                      # Обязательный параметр, по умолчанию 50 гигабайт.
      additionalTags:
        severity: critical                  # Необязательный параметр.
        environment: production            # Необязательный параметр.
  masterNodeGroup:
    replicas: 3
    instanceClass:
      flavorName: m1.large                   # Обязательный параметр.
      imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
      # Необязательный параметр. Если не указан – используется локальный диск.
      rootDiskSize: 50
      # Необязательный параметр, дополнительные группы безопасности.
      additionalSecurityGroups:
        - sec_group_1
        - sec_group_2
      additionalTags:
        severity: critical
        environment: production
      # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
      # (всегда используйте самый быстрый диск, предоставленный поставщиком).
      volumeTypeMap:

```

```

# Если указан rootDiskSize, этот тип тома также будет
# использоваться для главного корневого тома.
ru-1a: fast-ru-1a
ru-1b: fast-ru-1b
ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр, если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр, список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  zones:
    - ru-1a
    - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...

```

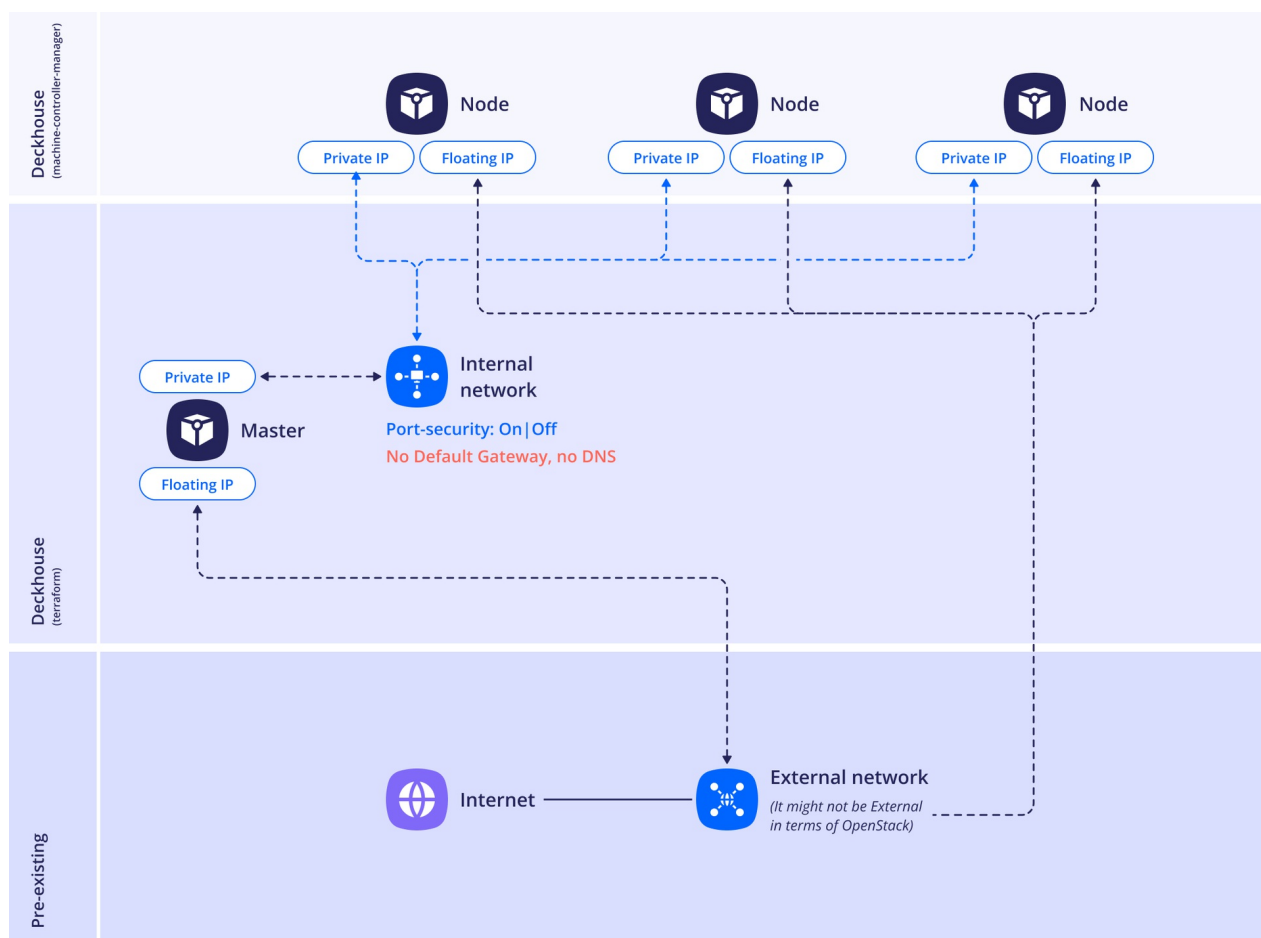
StandardWithNoRouter

Создается внутренняя сеть кластера без доступа в публичную сеть. Все узлы, включая master-узел, создаются с двумя интерфейсами: один — в публичную сеть, другой — во внутреннюю сеть. Данная схема размещения должна использоваться, если необходимо, чтобы все узлы кластера были доступны напрямую.

i

Внимание. В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal Loadbalancer, у которого Virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.

Внимание. В данной конфигурации необходимо явно указывать название внутренней сети в `additionalNetworks` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

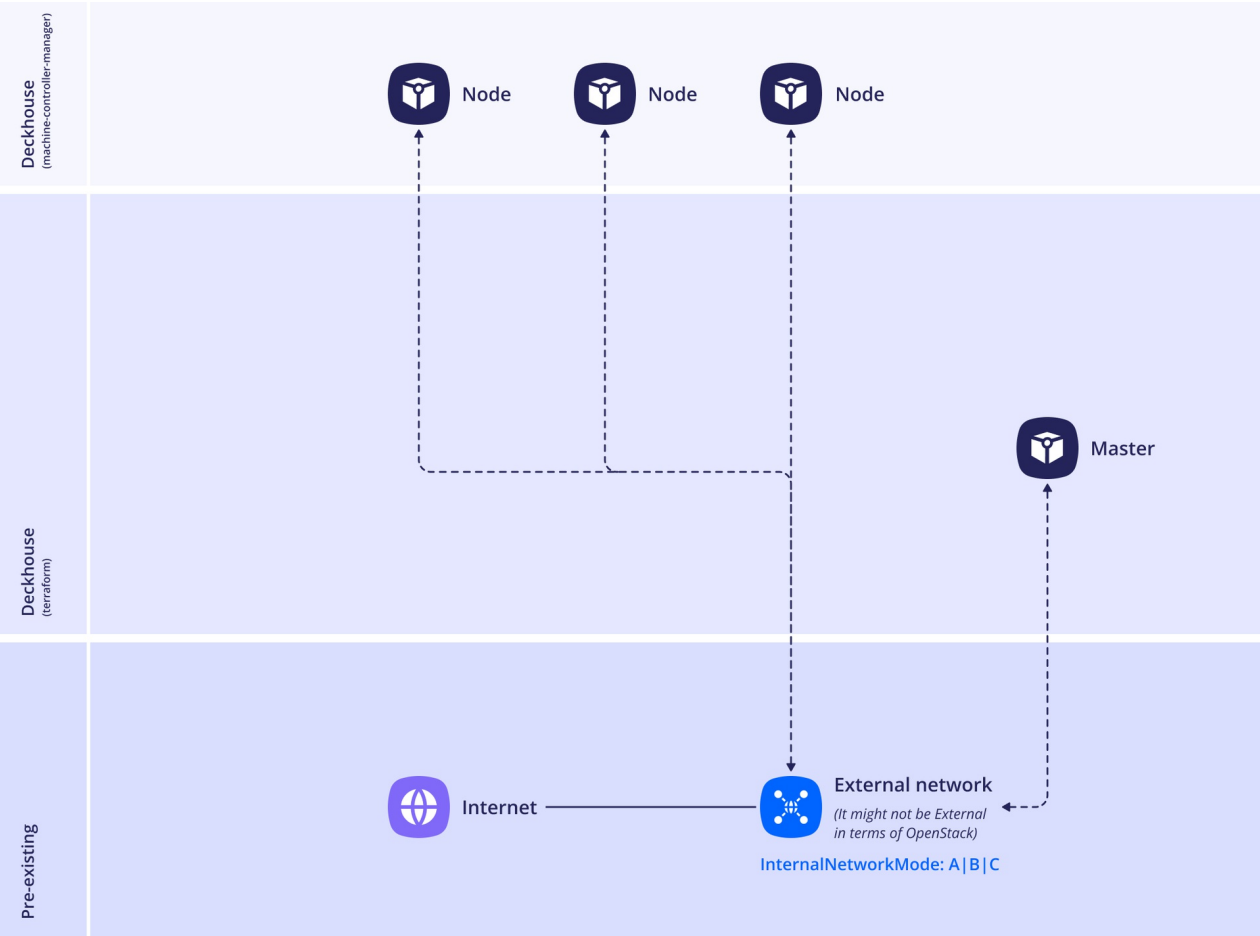
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24      # Обязательный параметр.
  externalNetworkName: ext-net              # Обязательный параметр.
  # Необязательный параметр, указывает, включен ли DHCP в указанной внешней сети
  # (по умолчанию true).
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию true.
  internalNetworkSecurity: true|false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64    # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                   # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64   # Обязательный параметр.
    # Необязательный параметр. Если не указан — используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, шлюз этой сети будет использован как шлюз по умолчанию.
    # Совпадает с cloud.prefix в ресурсе ClusterConfiguration.
    mainNetwork: kube
    additionalNetworks:                    # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPPools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Требуется, если указан параметр rootDiskSize. Карта типов томов для главного корневого тома
  # узла.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Simple

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

i **Внимание.** В данной конфигурации не поддерживается LoadBalancer. Это связано с тем, что в OpenStack нельзя заказать Floating IP для сети без роутера, соответственно, нельзя заказать балансировщик с Floating IP. Если заказывать internal loadbalancer, у которого virtual IP создается в публичной сети, он все равно доступен только с узлов кластера.



Пример конфигурации схемы размещения:

```

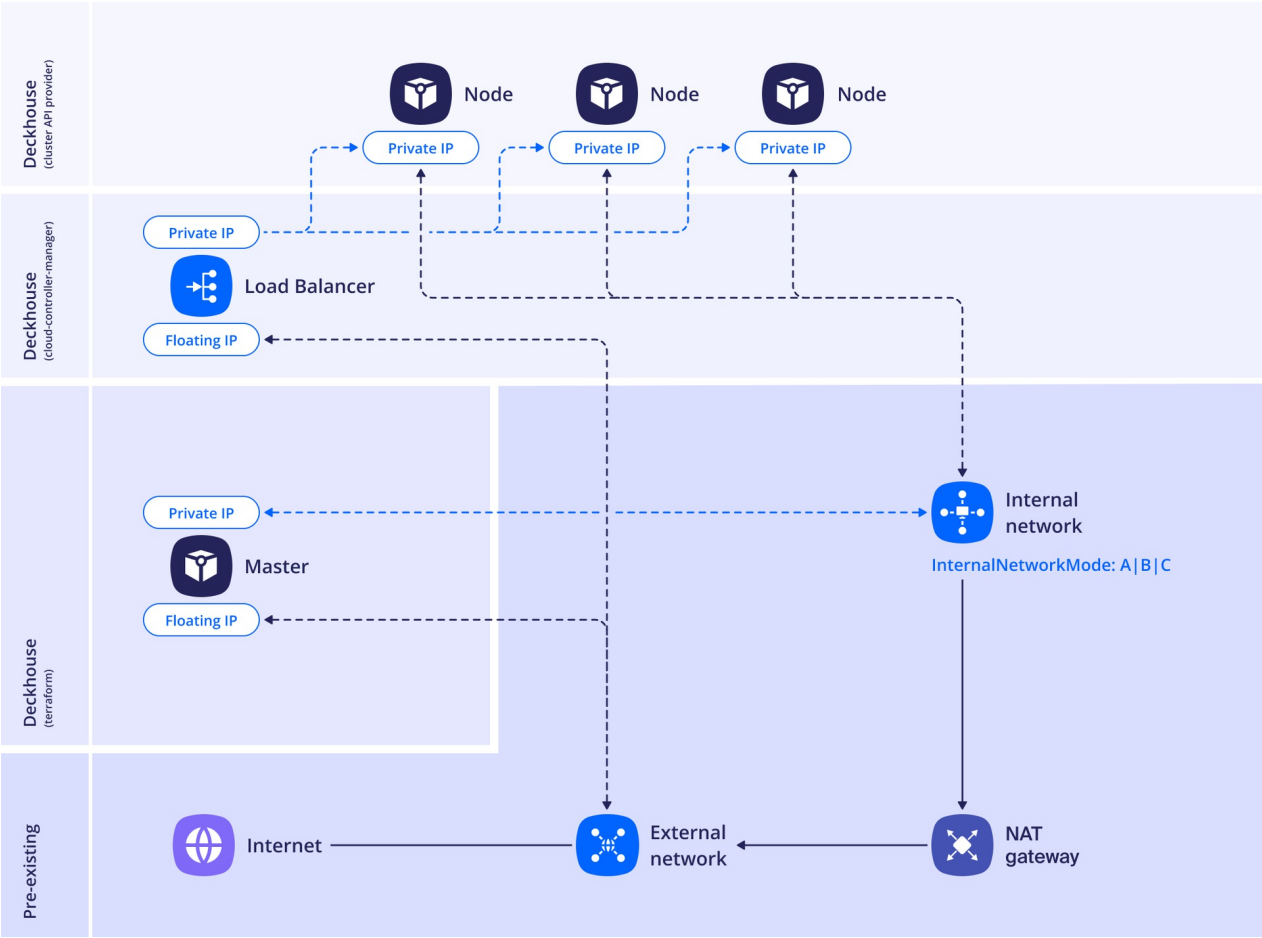
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net          # Обязательный параметр.
  # Необязательный параметр, по умолчанию true.
  externalNetworkDHCP: false
  # Необязательный параметр, по умолчанию VXLAN, также может быть DirectRouting
  # или DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Необязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр, сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks:                # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

SimpleWithInternalNetwork

Master-узел и узлы кластера подключаются к существующей сети. Данная схема размещения может понадобиться, если необходимо объединить кластер Kubernetes с уже имеющимися виртуальными машинами.

Внимание. В данной схеме размещения не происходит управление `SecurityGroups`, а подразумевается, что они были ранее созданы. Для настройки политик безопасности необходимо явно указывать `additionalSecurityGroups` в `OpenStackClusterConfiguration` для `masterNodeGroup` и других `nodeGroups`, а также `additionalSecurityGroups` при создании `OpenStackInstanceClass` в кластере.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Обязательный параметр, все узлы кластера должны находиться в одной подсети.
  internalSubnetName: pivot-standard
  # Необязательный параметр, по умолчанию DirectRoutingWithPortSecurityEnabled,
  # также может быть DirectRouting или VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Необязательный параметр. Если задан, будет использоваться для конфигурации
  # балансировщика нагрузки по умолчанию и для главного плавающего IP-адреса.
  externalNetworkName: ext-net
  # Необязательный параметр, по умолчанию true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр.
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 50
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  # Обязательный параметр. Карта типов томов для сертификатов etcd и Kubernetes
  # (всегда используйте самый быстрый диск, предоставляемый поставщиком).
  volumeTypeMap:
    # Если указан rootDiskSize, этот тип тома также будет
    # использоваться для главного корневого тома.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small # Обязательный параметр.
    imageName: ubuntu-18-04-cloud-amd64 # Обязательный параметр
    # Необязательный параметр. Если не указан – используется локальный диск.
    rootDiskSize: 20
    # Обязательный параметр, по умолчанию false. Определяет, требуется ли
    # конфигурационный диск во время процесса начальной загрузки виртуальной
    # машины. Это необходимо, если в сети нет DHCP, который используется в
    # качестве шлюза по умолчанию.
    configDrive: false
    # Обязательный параметр. Сеть будет использована как шлюз по умолчанию.
    # Совпадает с названием заранее созданной сети.
    mainNetwork: kube
    additionalNetworks: # Необязательный параметр.
      - office
      - shared
    # Необязательный параметр. Если существуют сети с отключенной защитой
    # портов, необходимо указать их имена.
    networksWithSecurityDisabled:
      - office
    # Необязательный параметр. Список сетевых пулов, в которых можно заказать
    # плавающие IP-адреса.
    floatingIPpools:
      - public
      - shared
    # Необязательный параметр, дополнительные группы безопасности.
    additionalSecurityGroups:
      - sec_group_1
      - sec_group_2
  sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...

```

Cloud provider — VMware Cloud Director

Модуль `cloud-provider-vcd` обеспечивает интеграцию Deckhouse Kubernetes Platform с VMware Cloud Director. Он предоставляет возможность модулю `node-manager` использовать ресурсы VCD при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-vcd`:

- Управление ресурсами VCD через `cloud-controller-manager`:
 - актуализирует метаданные виртуальных машин и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в VCD;
 - создаёт балансировщики нагрузки для Service с типом LoadBalancer. Для этого используется VMware NSX Advanced Load Balancer (Avi); поддержка доступна при использовании NSX-T.
- Заказ дисков через CSI-драйвер Named Disk (`named-disk.csi.cloud-director.vmware.com`), чтобы из кластера можно было заказывать PersistentVolume.
- Заказ CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-vcd`. Базовая инфраструктура заказывается в зависимости от схемы размещения.
- Заказ CloudEphemeral-узлов через Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе VCDInstanceClass.
- Регистрация в модуле `node-manager`, чтобы VCDInstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.



Для работы с API VCD версии ниже 37.2 модуль использует режим совместимости (legacy-компоненты).

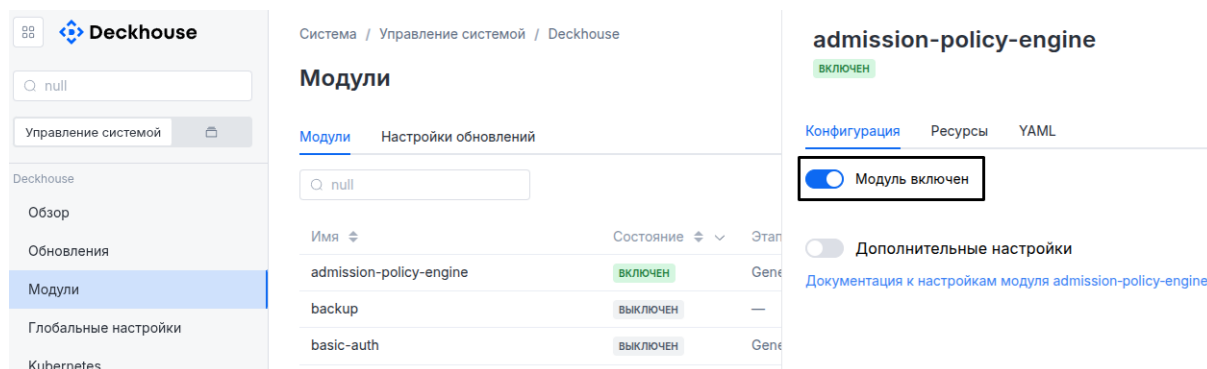
Настройки

Модуль автоматически включается для всех облачных кластеров, развернутых в VMware Cloud Director.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vcd`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-vcd`:

```
d8 system module enable cloud-provider-vcd
```

- **С помощью ModuleConfig `cloud-provider-vcd`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-vcd` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-vcd`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vcd
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vcd`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cloud-provider-vcd` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vcd`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-vcd` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-vcd
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-vcd`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-vcd`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vcd
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.storageClass**
объект

Автоматическое создание StorageClass'ов, которые есть в VCD (VMware Cloud Director).

■ settings.storageClass.default

Параметр устарел строка

Имя StorageClass'a, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- произвольный StorageClass в кластере, имеющий default-аннотацию;
- StorageClass, использующий default StorageProfile;
- первый (по алфавиту) StorageClass из тех, что создаются модулем.

i **Параметр устарел.** Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: ceph-ssd
```

■ **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) StorageClass'ов, которые не будут созданы в кластере.

Пример:

```
exclude:
- ".*-hdd"
- iscsi-fast
```

Настройки провайдера

i

Если control plane кластера размещен на виртуальных машинах или bare-metal-серверах, cloud-провайдер использует настройки модуля `cloud-provider-vcd` в конфигурации Deckhouse. Иначе, если control plane кластера размещен в облаке, cloud-провайдер использует структуру `VCDClusterConfiguration` для настройки.

VCDClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в VCD.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в запущенном кластере:

```
d8 system edit provider-cluster-configuration
```

i

После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    template: Org/Templates/ubuntu-focal-20.04
    sizingPolicy: 16cpu32ram
    storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

• **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`

- **bastion**
объект

Параметры конфигурации бастийон-хоста, используемого для обеспечения административного доступа к узлам кластера.

- **bastion.instanceClass**
объект

Обязательный параметр

Частичное содержимое полей VCDInstanceClass.

- **bastion.instanceClass.additionalMetadata**
объект

Пользовательские строковые пары ключ-значение, которые будут добавлены в раздел метаданных (Metadata) каждой виртуальной машины master-узлов. Это позволяет более эффективно организовывать и классифицировать виртуальные машины в вашей системе виртуализации.

Свойства из этого поля будут объединены с уже существующими метаданными, хранящимися в поле `metadata` в корне спецификации. В случае совпадения ключей значения из этого поля имеют приоритет.

- **bastion.instanceClass.mainNetworkIPAddress**
строка

IP-адрес бастийон-хоста в `mainNetwork`.

По умолчанию DHCP-клиент включен.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **bastion.instanceClass.placementPolicy**
строка

Политика размещения (PlacementPolicy), которая будет применяться к данному хосту.

Если политика размещения не указана, будет использована политика размещения по умолчанию для создания хостов.

Длина: `1..128`

- **bastion.instanceClass.rootDiskSizeGb**
целочисленный

Размер root-диска. Значение указывается в гигабайтах.

По умолчанию: `50`

Допустимые значения: `0 <= X`

Пример:

```
rootDiskSizeGb: 50
```

- **bastion.instanceClass.sizingPolicy**
строка

Обязательный параметр

Политика масштабирования (SizingPolicy), которая будет использоваться на данном хосте.

Если политика масштабирования не указана, будет использована политика масштабирования по умолчанию для создания хостов.

Длина: `1..128`

- **bastion.instanceClass.storageProfile**
строка

Обязательный параметр

Профиль хранения данных (StorageProfile), который будет использоваться на данном хосте.

Длина: `1..128`

- **bastion.instanceClass.template**

Обязательный параметр

Путь к шаблону OVA, который будет использоваться.

Путь состоит из имени организации (опционально), каталога и имени или идентификатора шаблона. Если используется общий каталог, имя организации указывать обязательно (оно может отличаться от значения `provider.organization`).

Если в нескольких каталогах с одинаковым названием есть шаблоны с одинаковыми именами, необходимо указать идентификатор шаблона.

Идентификатор шаблона можно найти в URL-адресе при открытии шаблона в пользовательском интерфейсе VCD. Например, если URL-адрес `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, то идентификатор шаблона будет `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Длина: `1..128`

- **createDefaultFirewallRules**
булевый

Если включено, настраивает следующие правила брандмауэра по умолчанию для основной сети кластера (mainNetwork):

- Разрешает все исходящие соединения из mainNetwork.
- Разрешает все входящие ICMP-пакеты в mainNetwork.
- Разрешает все входящие TCP-соединения в mainNetwork на порт 22 (SSH).
- Разрешает все входящие TCP и UDP-соединения в mainNetwork для NodePorts в диапазоне 30000-32767.

Внимание! Использование этой опции совместно с Edge Gateway типа NSX-T подразумевает, что к одному Edge Gateway может быть привязан только один кластер, поскольку в этом случае все существующие правила брандмауэра будут перезаписаны.

По умолчанию: `false`

- **edgeGateway**
объект

Свойства VMware Cloud Director Edge Gateway (принадлежащее Virtual Data Center).

- **edgeGateway.NSX-V**
объект

Дополнительные параметры, необходимые, если `type` равно NSX-V.

- **edgeGateway.NSX-V.externalNetworkName**
строка

Имя внешней сети для создания правила DNAT для мастер-узла. Внешняя сеть — это сеть, подключённая к Edge Gateway в разделе `Gateway Interface` и имеющая внешний IP-адрес.

Длина: `1..128`

- **edgeGateway.NSX-V.externalNetworkType**
строка

Тип внешней сети для создания правила DNAT для мастер-узла (один из “org” или “ext”).

Допустимые значения: `org`, `ext`

- **edgeGateway.externalIP**
строка

Обязательный параметр

IP-адрес, доступный из внешних сетей, через который входящие SSH-соединения перенаправляются на мастер-узел с использованием DNAT.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **edgeGateway.externalPort**
целочисленный

Внешний порт, сопоставленный правилом DNAT для перенаправления входящих SSH соединений на мастер-узел.

По умолчанию: `22`

Допустимые значения: `1 <= X <= 65535`

Пример:

```
externalPort: 22
```

- **edgeGateway.name**
строка

Обязательный параметр

Имя VMware Cloud Director Edge Gateway.

Длина: `1..128`

- **edgeGateway.type**
строка

Обязательный параметр

Тип платформы сетевой виртуализации, обеспечивающей работу VMware Cloud Director Edge Gateway (один из “NSX-V” или “NSX-T”). Тип платформы можно узнать у администратора.

Допустимые значения: `NSX-V`, `NSX-T`

- **internalNetworkCIDR**

строка

Адресация для внутренней сети узлов кластера.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkDHCPPoolStartAddress**

целочисленный

Задаёт номер адреса в внутренней сети, с которого начинается пул DHCP-адресов. Адреса с номерами ниже этого значения не будут выдаваться автоматически по DHCP и доступны для ручного назначения. Если значение не указано, по умолчанию используется 30.

Например, в сети 192.168.100.128/27 значение 10 означает, что пул DHCP начнется с IP-адреса 192.168.100.138. Пул закончится на адресе, который непосредственно предшествует широковещательному (192.168.100.159), то есть на последнем пригодном для назначения адресе.

Допустимые значения: `2 <= X`

- **internalNetworkDNSServers**

массив строк

Список адресов рекурсивных DNS серверов.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **kind**

строка

Обязательный параметр

Допустимые значения: `VCDClusterConfiguration`

- **layout**

строка

Обязательный параметр

Способ размещения ресурсов в облаке.

Подробнее о возможных вариантах размещения.

- **legacyMode**

булевый

Режим поддержки устаревших версий API VCD.

Если задано значение `true`, будет использоваться версия API ниже `37.2`.

Параметр имеет эффект только при установке кластера. В работающем кластере необходимость включения режима поддержки устаревших версий API определяется автоматически.

По умолчанию: `false`

- **mainNetwork**

строка

Обязательный параметр

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Пример:

```
mainNetwork: internal
```

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `VCDInstanceClass`.

- **masterNodeGroup.instanceClass.additionalMetadata**

объект

Пользовательские строковые пары ключ-значение, которые будут добавлены в раздел метаданных (Metadata) каждой виртуальной машины master-узлов. Это позволяет более эффективно организовывать и классифицировать виртуальные машины в вашей системе виртуализации.

Свойства из этого поля будут объединены с уже существующими метаданными, хранящимися в поле `metadata` в корне спецификации. В случае совпадения ключей значения из этого поля имеют приоритет.

■ **masterNodeGroup.instanceClass.affinityRule**
объект

Определяет правило размещения, применяемое к виртуальным машинам master-узлов внутри кластера. Такие правила определяют, как узлы располагаются друг относительно друга на уровне гипервизоров, что может повысить отказоустойчивость и контролировать распределение рабочих нагрузок. Если не указано, правило размещения применяться не будет.

■ **masterNodeGroup.instanceClass.affinityRule.polarity**
строка

Обязательный параметр

Полярность правила размещения. Должно быть либо `Affinity`, либо `AntiAffinity`. `Affinity` означает, что узлы должны быть размещены на одном хосте, в то время как `AntiAffinity` означает, что они должны быть размещены на разных хостах.

Допустимые значения: `Affinity`, `AntiAffinity`

■ **masterNodeGroup.instanceClass.affinityRule.required**
булевый

Указывает, является ли правило размещения обязательным (`true`) или предпочтительным (`false`). Если установлено в `true`, правило должно быть соблюдено; если в `false`, правило применяется, если это возможно, но не является строго обязательным.

По умолчанию: `false`

■ **masterNodeGroup.instanceClass.etcdDiskSizeGb**
целочисленный

Размер диска etcd. Значение указывается в гигабайтах.

По умолчанию: `10`

Пример:

```
etcdDiskSizeGb: 10
```

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses**
массив строк

Список статических IP-адресов (с префиксом CIDR), последовательно распределенных по узлам в `mainNetwork`.

По умолчанию DHCP-клиент включен.

Пример:

```
mainNetworkIPAddresses:
- 10.1.14.20
- 10.1.4.21
```

■ **Элемент массива**
строка

IP-адрес с префиксом CIDR.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

■ **masterNodeGroup.instanceClass.placementPolicy**
строка

PlacementPolicy, которая будет использоваться на данном компьютере.

Если PlacementPolicy не указана, для создания узлов будет использоваться PlacementPolicy по умолчанию.

Пример:

```
placementPolicy: Hypervisor1
```

■ **masterNodeGroup.instanceClass.rootDiskSizeGb**
целочисленный

Размер root-диска. Значение указывается в гигабайтах.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

- **masterNodeGroup.instanceClass.sizingPolicy**

строка

Обязательный параметр

SizingPolicy, которая будет использоваться на данном компьютере.

Если SizingPolicy не указана, для создания узлов будет использоваться SizingPolicy по умолчанию.

Пример:

```
sizingPolicy: 4Cpu8Memory
```

- **masterNodeGroup.instanceClass.storageProfile**

строка

Обязательный параметр

StorageProfile, который будет использоваться на данном компьютере.

Пример:

```
storageProfile: ssd
```

- **masterNodeGroup.instanceClass.template**

строка

Обязательный параметр

Путь к шаблону OVA, который будет использоваться.

Состоит из организации (опционально), каталога и названия или идентификатора шаблона. Если используется общий каталог, имя организации указывать обязательно (оно может отличаться от значения `provider.organization`).

Если в нескольких каталогах с одинаковым названием есть шаблоны с одинаковым именем, следует указать идентификатор шаблона.

Идентификатор шаблона можно найти в URL-адресе при открытии шаблона в пользовательском интерфейсе vCD. Например, если URL-адрес `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, то идентификатор шаблона будет `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Примеры:

```
template: Linux/ubuntu2204-cloud-ova
```

```
template: MyOrg/Linux/ubuntu2204-cloud-ova
```

```
template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

```
template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **metadata**

объект

Пользовательские метаданные, которые будут сохранены в объектах кластера, таких как виртуальные машины, диски и сеть. Это позволяет более эффективно организовывать и классифицировать ресурсы в вашей системе виртуализации.

Очистка полей `metadata` и `additionalMetadata` при наличии в них ключей приведет к прекращению отслеживания всех метаданных, за исключением узлов типа `CloudEphemeral`. В таком случае удаление метаданных возможно только вручную.



Внимание. Изменение метаданных приведет к пересозданию `CloudEphemeral`-узлов.

- **nodeGroups**

массив объектов

Массив дополнительных `NodeGroup` для создания статических узлов (например, для выделенных `frontend`-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей VCDInstanceClass.

- **nodeGroups.instanceClass.additionalMetadata**

объект

Пользовательские строковые пары ключ-значение, которые будут добавлены в раздел метаданных (Metadata) каждой виртуальной машины master-узлов. Это позволяет более эффективно организовывать и классифицировать виртуальные машины в вашей системе виртуализации.

Свойства из этого поля будут объединены с уже существующими метаданными, хранящимися в поле `metadata` в корне спецификации. В случае совпадения ключей значения из этого поля имеют приоритет.

- **nodeGroups.instanceClass.affinityRule**

объект

Определяет правило размещения, применяемое к виртуальным машинам узлов данной NodeGroup внутри кластера. Такие правила определяют, как узлы располагаются друг относительно друга на уровне гипервизоров, что может повысить отказоустойчивость и контролировать распределение рабочих нагрузок. Если не указано, правило размещения применяться не будет.

- **nodeGroups.instanceClass.affinityRule.polarity**

строка

Обязательный параметр

Полярность правила размещения. Должно быть либо `Affinity`, либо `AntiAffinity`. `Affinity` означает, что узлы должны быть размещены на одном хосте, в то время как `AntiAffinity` означает, что они должны быть размещены на разных хостах.

Допустимые значения: `Affinity`, `AntiAffinity`

- **nodeGroups.instanceClass.affinityRule.required**

булевый

Указывает, является ли правило размещения обязательным (`true`) или предпочтительным (`false`). Если установлено в `true`, правило должно быть соблюдено; если в `false`, правило применяется, если это возможно, но не является строго обязательным.

По умолчанию: `false`

- **nodeGroups.instanceClass.mainNetworkIPAddresses**

массив строк

Список статических IP-адресов (с префиксом CIDR), последовательно распределенных по узлам в `mainNetwork`.

По умолчанию DHCP-клиент включен.

Пример:

```
mainNetworkIPAddresses:
- 10.1.14.20
- 10.1.4.21
```

- **Элемент массива**

строка

IP-адрес с префиксом CIDR.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **nodeGroups.instanceClass.placementPolicy**

строка

PlacementPolicy, которая будет использоваться на данном компьютере.

Если PlacementPolicy не указана, для создания узлов будет использоваться PlacementPolicy по умолчанию.

Пример:

```
placementPolicy: Hypervisor1
```

- **nodeGroups.instanceClass.rootDiskSizeGb**

целочисленный

Размер root-диска. Значение указывается в гигабайтах.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

- **nodeGroups.instanceClass.sizingPolicy**

строка

Обязательный параметр

SizingPolicy, которая будет использоваться на данном компьютере.

Если SizingPolicy не указана, для создания узлов будет использоваться SizingPolicy по умолчанию.

Пример:

```
sizingPolicy: 4Cpu8Memory
```

- **nodeGroups.instanceClass.storageProfile**

строка

Обязательный параметр

StorageProfile, который будет использоваться на данном компьютере.

Пример:

```
storageProfile: ssd
```

- **nodeGroups.instanceClass.template**

строка

Обязательный параметр

Путь к шаблону OVA, который будет использоваться.

Состоит из организации (опционально), каталога и названия или идентификатора шаблона. Если используется общий каталог, имя организации указывать обязательно (оно может отличаться от значения `provider.organization`).

Если в нескольких каталогах с одинаковым названием есть шаблоны с одинаковым именем, следует указать идентификатор шаблона.

Идентификатор шаблона можно найти в URL-адресе при открытии шаблона в пользовательском интерфейсе vCD. Например, если URL-адрес

`https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, то идентификатор шаблона будет `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Примеры:

```
template: Linux/ubuntu2204-cloud-ova
```

```
template: MyOrg/Linux/ubuntu2204-cloud-ova
```

```
template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

```
template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **organization**

строка

Обязательный параметр

Имя VMware Cloud Director Organization.

- **provider**

объект

Обязательный параметр

Параметры для подключения к VCD.

- **provider.apiToken**

строка

Токен для аутентификации.

i **Внимание.** При использовании `apiToken` необходимо оставить `username` и `password` пустыми.

- **provider.insecure**

булевый

Устанавливается в `true`, VCD имеет self-signed-сертификат.

По умолчанию: `false`

- **provider.password**

строка

Пароль пользователя.

- **provider.server**

строка

Обязательный параметр

Хост или IP-адрес VCD-сервера.

- **provider.username**

строка

Имя пользователя с полными правами на проект.

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **virtualApplicationName**

строка

Обязательный параметр

Имя VMware Cloud Director Virtual Application (принадлежащее Virtual Data Center).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **virtualDataCenter**

строка

Обязательный параметр

Имя VMware Cloud Director Virtual Data Center (принадлежащее Organization).

Подготовка окружения.

Список необходимых ресурсов VCD

- Organization
- VirtualDataCenter
- vApp (при схеме размещения `Standard`)
- StoragePolicy
- SizingPolicy
- Network (при схеме размещения `Standard`)
- EdgeRouter
- Catalog

Ресурсы Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter и Catalog должны быть предоставлены вашим поставщиком услуг VMware Cloud Director.



В каждом VDC (Virtual Data Center) должен быть настроен Edge Gateway, сеть кластера должна быть подключена к нему.

Network (внутренняя сеть) может быть настроена вашим поставщиком услуг VMware Cloud Director, либо вы можете настроить ее самостоятельно. При выборе схемы размещения `WithNAT` сеть создается автоматически. Далее описан способ самостоятельной настройки внутренней сети.

Права пользователя

Для доступа к API VMware Cloud Director достаточно одного из следующих вариантов:

- роль `Organization Administrator` с правом `Preserve All ExtraConfig Elements During OVF Import and Export`. Такой вариант не требует настройки отдельной роли, но предоставляет пользователю полный административный доступ к организации;
- пользовательская роль с набором прав, перечисленным ниже. Этого набора достаточно для создания кластера, заказа узлов, включая CloudEphemeral со статической адресацией и DHCP, а также для работы с CSI-дисками и StorageClass без предоставления дополнительных административных прав.

В VMware Cloud Director права пользователя складываются из роли и из **Rights Bundle** — набора прав, который публикуется для организации. Право `Preserve All ExtraConfig Elements During OVF Import and Export` нужно добавить и в роль, и в Rights Bundle организации.



Без права `Preserve All ExtraConfig Elements During OVF Import and Export` в роли и в Rights Bundle организации `userdata` не передаётся в виртуальные машины эфемерных узлов.

Чтобы создать пользовательскую роль, включите права, как показано ниже. Названия прав приведены как в интерфейсе VCD.

1. В разделах «ACCESS CONTROL» и «ADMINISTRATION» включите следующие права:

- Organization → View: `View Organization Administrative Details`, `View vApp ACL`;
- User → Manage: `Manage user's own API token`;
- General → View: `View Certificates Library`, `Administrator View`, `View Quota Policy Capabilities`;
- General → Manage: `Administrator Control`.

Name	View (47 / 121)	Manage (44 / 191)
ACCESS CONTROL		
Organization	View (27 / 7)	Manage (0 / 11)
	- Allow Access to All Organization VDCs - View Catalog ACL - View Access Control List of Organization VDCs ✓ View Organization Administrative Details - View Organization Associations - View OAuth Settings ✓ View vApp ACL	- Edit Organization Network Properties - Edit Access Control List of Organization VDCs - Edit Organization Associations - Edit Federation Settings - Edit Leases Policy - Edit Organization OAuth Settings - Edit Password Policy - Edit Organization Properties - Edit Quotas Policy - Edit SMTP Settings - Implicitly Import User/Group from IdP while Editing VDC ACL
Role		0 / 1
User	View (0 / 1)	Manage (1 / 3)
	View Users and Groups	✓ Manage user's own API token - Manage all users' API tokens - Manage Users and Groups
ADMINISTRATION		
General	View (17 / 12)	Manage (1 / 11)
	- View Advisories ✓ View Certificates Library - View General ACL ✓ Administrator View - View Error Details - View Shared Compute Policy ✓ View Quota Policy Capabilities - View Quota Policy - Test Connection - View task information - View Trusted Certificates - View vGPU Profile Consumption	- Manage Advisories - Manage Certificates Library - Manage General ACL ✓ Administrator Control - Send Notification - Publish Compute Policy - Migrate Storage - Manage Quota Policy - Resume / Cancel / Fail a Task - Update Task - Manage Trusted Certificates

2. В разделе «COMPUTE» → «Organization VDC» включите все права View и следующие права Manage:

- Edit Disk IOPS;
- Edit Tenant Kubernetes Policy;
- Change Owner;
- Create a Disk;
- Delete a Disk;
- Edit Disk Properties;
- Move a Disk;
- Create a Shared Disk.

В разделе «Provider VDC» → View включите **Limited Provider VDC Storage Policy View** и **Limited Provider VDC View**:

COMPUTE		
Container Applications	0 / 1	0 / 1
Kubernetes Namespace	0 / 2	0 / 1
Organization VDC	View (9 / 9)	Manage (7 / 19)
	✓ Limited Admin View ✓ View Compute Policies for an Organization VDC ✓ View Disk IOPS ✓ View Disk Encryption Status ✓ View Disk Properties ✓ View Organization VDC Storage Policy Capabilities ✓ View Organization VDC Extended Properties ✓ View Organization VDCs ✓ View Organization VDC Memory and CPU Reservation	✓ Edit Disk IOPS - Edit Tenant Kubernetes Policy ✓ Change Owner ✓ Create a Disk ✓ Delete a Disk ✓ Edit Disk Properties ✓ Move a Disk ✓ Create a Shared Disk - Edit Organization VDC Storage Policy - Disable Organization VDC Storage Policy - Remove Organization VDC Storage Policy - Set Default Storage Policy - Create Organization VDC - Delete Organization VDCs - Edit Organization VDC Name and Description - Enable / Disable Organization VDC - Edit Organization VDC Extended Properties - Manage Firewall - Edit VM-VM Affinity Rule
Provider VDC	View (2 / 2)	
	✓ Limited Provider VDC Storage Policy View ✓ Limited Provider VDC View	

3. В разделе «COMPUTE» → «vApp» включите права, как на скриншоте. Убедитесь, что включено **Preserve All ExtraConfig Elements During OVF Import and Export**. В проверенной роли среди View включены **View Encryption Status of VMs and VM's disks** и **View VM metrics**; **View vApp Shadow VMs** и остальные варианты Preserve ExtraConfig не включены:

Provider VDC	View (2/2)	
	✓ Limited Provider VDC Storage Policy View ✓ Limited Provider VDC View	
vApp	View (2/3)	Manage (23/29)
	- View vApp Shadow VMs ✓ View Encryption Status of VMs and VM's disks ✓ View VM metrics	- Change vApp Template Owner ✓ Preserve All ExtraConfig Elements During OVF Import and Export - Preserve Ethernet-Coalescing ExtraConfig Elements During OVF Import and Export - Preserve Latency ExtraConfig Elements During OVF Import and Export - Preserve ExtraConfig Elements During OVF Import and Export if they match patterns specified by the system administrator in the 'vapp.allowed.extra.config' configuration property - Preserve NUMA Node Affinity ExtraConfig Elements During OVF Import and Export - Change Owner ✓ Copy a vApp ✓ Create / Reconfigure a vApp ✓ Delete a vApp ✓ Download a vApp ✓ Edit vApp Properties ✓ Edit VM Compute Policy ✓ Edit VM CPU ✓ Edit VM CPU and Memory Reservation / Limit / Shares in all VDC types ✓ Edit VM Hard Disk ✓ Edit VM Memory ✓ Edit VM Network ✓ Edit VM Properties ✓ Enter / Exit vApp Maintenance Mode ✓ Manage VM Password Settings ✓ Start / Stop / Suspend / Reset a vApp ✓ Share a vApp ✓ Create / Revert / Remove a Snapshot ✓ Upload a vApp ✓ Access to VM Console ✓ Edit / View VM Boot Options ✓ Migrate / Force Undeploy / Relocate / Consolidate vApp VMs ✓ Allow metadata mapping domain to vCenter

4. В разделе «LIBRARIES» включите следующие права:

- Catalog → View: View Private and Shared Catalogs within Current Organization, View Shared Catalogs from Other Organizations (чтобы использовать образы из общих каталогов);
- Catalog → Manage: Add a vApp from My Cloud;
- Catalog Item → View: View vApp Templates / Media;
- Catalog Item → Manage: Copy / Move a vApp Template / Media, Edit vApp Template / Media Properties, Add to My Cloud.

В разделе «VMware Cloud Director Extension» → View включите View Tenant Portal Plugin Information:

VMware Cloud Director Extension	View (1/2)	Manage (0/1)
	- View Access Control List ✓ View Tenant Portal Plugin Information	Edit Access Control List
INFRASTRUCTURE		
Load Balancer	0/2	0/1
SDDC	0/1	0/4
vCenter		0/1
LIBRARIES		
Catalog	View (2/3)	Manage (1/8)
	- View Catalog vApp Shadow VMs ✓ View Private and Shared Catalogs within Current Organization ✓ View Shared Catalogs from Other Organizations	✓ Add a vApp from My Cloud - Change Owner - Allow External Publishing / Subscriptions for the Catalogs - Create / Delete a Catalog - Edit Catalog Properties - Share a Catalog to Other Organizations - Share a Catalog to Users / Groups within Current Organization - Allow VCSP Publish Subscribe Caching for Catalog
Catalog Item	View (1/1)	Manage (2/5)
	✓ View vApp Templates / Media	✓ Copy / Move a vApp Template / Media - Create / Upload a vApp Template / Media ✓ Edit vApp Template / Media Properties ✓ Add to My Cloud - Enable vApp Template / Media Download
NETWORKING		

5. В разделе «NETWORKING» включите следующие права:

- Edge Gateway → View: View Gateway;
- Edge Gateway Services → View: все пункты;
- Edge Gateway Services → Manage: Load Balancer Configure, NAT Configure;
- IP Spaces → View: View IP Spaces;
- IP Spaces → Manage: Allocate IP Spaces, Manage IP Spaces;
- Organization VDC Network → View: View Properties;
- Organization VDC Network → Manage: Edit Properties (чтобы создавать сети в организации).

NETWORKING		
> Distributed Firewall	☐ 0/1	☐ 0/2
> Edge Cluster	☐ 0/1	
▼ Edge Gateway	☒ View 1/1	☐ Manage 0/8
	✓ View Gateway	- Configure Syslog Server - Configure System Logging - Convert to Advanced Gateway - Create Gateway - Delete Gateway - Enable Distributed Routing - Update Gateway - Update Properties of Gateway
▼ Edge Gateway Services	☒ View 14/14	☐ Manage 2/15
	✓ BGP Routing View Only ✓ DHCP View Only ✓ Edge Gateway DNS View Only ✓ Firewall View Only ✓ IPSEC VPN View Only ✓ L2 VPN View Only ✓ Load Balancer View Only ✓ NAT View Only ✓ OSPF Routing View Only ✓ Remote Access View Only ✓ View Route Advertisement ✓ View SLAAC Profile ✓ SSL VPN View Only ✓ Static Routing View Only	- BGP Routing Configure - DHCP Configure - Edge Gateway DNS Configure - ECMP Routing Configure - Firewall Configure - IPSEC VPN Configure - L2 VPN Configure ✓ Load Balancer Configure ✓ NAT Configure - OSPF Routing Configure - Remote Access Configure - Configure Route Advertisement - Configure SLAAC Profile - SSL VPN Configure - Static Routing Configure
▼ IP Spaces	☒ View 1/1	☒ Manage 2/2
	✓ View IP Spaces	✓ Allocate IP Spaces ✓ Manage IP Spaces
▼ Organization VDC Network	☒ View 1/1	☐ Manage 1/3
	✓ View Properties	✓ Edit Properties - Import Network - Manage Manual IP Reservation

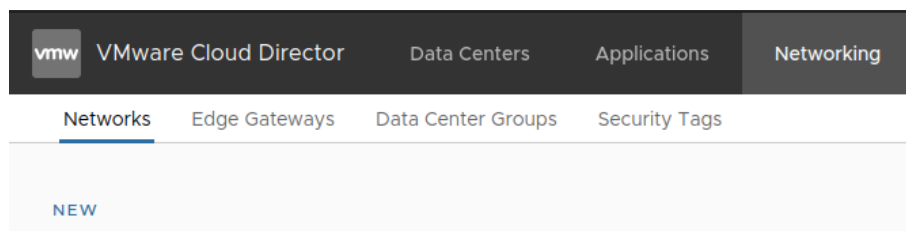
6. В разделе «Provider Gateway Services» включите все права View. В разделе «KubeClusterExtension» включите все права View и Manage:

> Network Pool	☐ 0/1	
▼ Provider Gateway Services	☒ View 6/6	☐ Manage 0/4
	✓ View Limited Provider Gateway BGP ✓ View Provider Gateway Firewall ✓ View Provider Gateway IPSec VPN ✓ View Provider Gateway NAT ✓ View Provider Gateway Routing (Simple View) ✓ View Limited Provider Gateway	- Manage Limited Provider Gateway BGP - Manage Provider Gateway Firewall - Manage Provider Gateway IPSec VPN - Manage Provider Gateway NAT
> Provider Network	☐ 0/1	
> Segment Profile Templates	☐ 0/1	
> VDC Group	☐ 0/1	☐ 0/3
OTHER		
> Annotation	☐ 0/2	☐ 0/3
> Branding UI Theme	☐ 0/2	☐ 0/3
> ContainerRegistryCredential	☐ 0/2	☐ 0/3
▼ KubeClusterExtension	☒ View 2/2	☒ Manage 3/3
	✓ Administrator View ✓ View	✓ Administrator Full Control ✓ Full Control ✓ Edit

Добавление сети

i Инструкция актуальна только для схемы размещения **Standard**.

1. Перейдите на вкладку «Networking» и нажмите «NEW»:



2. Выберите необходимый Data Center:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Scope

Organization Virtual Data Center

Provides connectivity for VMs in the selected VDC only

	Name	Allocation Model	Organization
☒	flant-df-lab_DC1	Pay-As-You-Go	flant-df-lab_DC1

1 - 1 of 1 data center(s)

CANCEL

NEXT

3. На этапе «Network type» выберите «Routed»:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Network Type

Select the type of network that you are about to create

☒ Routed

This type of network provides controlled access to machines and networks outside of the VDC or VDC Group through an edge gateway.

☐ Isolated

This type of network provides a fully isolated environment, which is accessible only by this organization VDC or VDC Group.

CANCEL

PREVIOUS

NEXT

4. Присоедините EdgeRouter к сети:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Edge Connection

Name	External Networks	Org VDC Networks
EDGE-flant-df-lab_DC1	1	2

1 - 1 of 1 Edge Gateway(s)

Distributed Routing

Guest VLAN Allowed

CANCEL

PREVIOUS

NEXT

5. Укажите имя сети и CIDR:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

General

Name *

internal

Description

Dual-Stack Mode

Gateway CIDR *

192.168.199.1/24

Shared

CANCEL

PREVIOUS

NEXT

6. Не добавляйте «Static IP Pools», поскольку будет использоваться DHCP:

Страница 1991

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

Static IP Pools

Gateway CIDR192.168.199.1/24 ⓘ

Static IP Pools

Enter an IP range (format: 192.168.1.2 - 192.168.1.100)

ADD

MODIFY

REMOVE

Total IP addresses: 0

CANCEL

PREVIOUS

NEXT

7. Укажите адреса DNS-серверов:

New Organization VDC Network

1 Scope

2 Network Type

3 Edge Connection

4 General

5 Static IP Pools

6 DNS

7 Ready to Complete

DNS

Primary DNS8.8.8.8

Secondary DNS8.8.4.4

DNS suffix

CANCEL

PREVIOUS

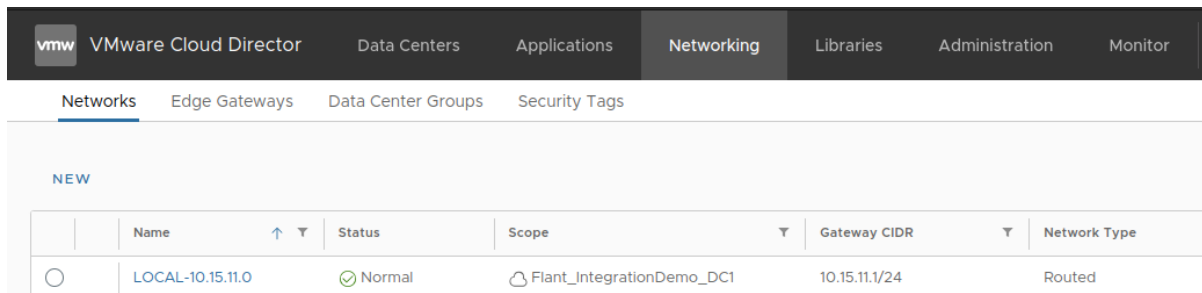
NEXT

i Инструкция актуальна только для схемы размещения **Standard**.

Для динамического заказа узлов включите DHCP-сервер для внутренней сети.

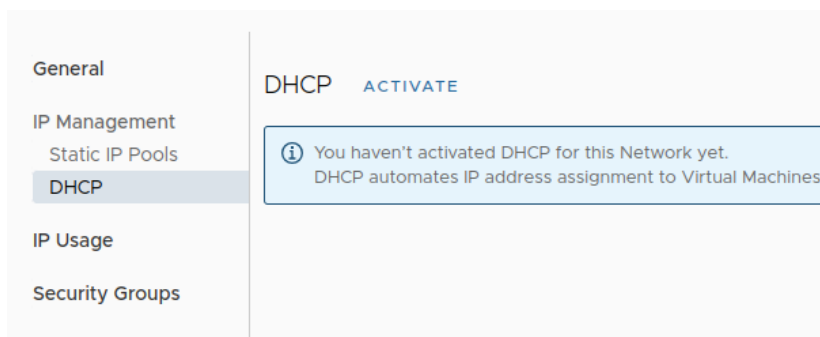
i Рекомендуем выделить начало диапазона адресов сети на системные нагрузки (control plane, frontend-узлы, системные узлы), а остальное выделить на DHCP-пул. Например, для сети по маске /24 будет достаточно выделения 20 адресов под системные нагрузки.

1. Перейдите на вкладку «Networking» и откройте созданную сеть:



VMware Cloud Director						
	Data Centers	Applications	Networking	Libraries	Administration	Monitor
Networks Edge Gateways Data Center Groups Security Tags						
NEW						
	Name	Status	Scope	Gateway CIDR	Network Type	
☐	LOCAL-10.15.11.0	Normal	Flant_IntegrationDemo_DC1	10.15.11.1/24	Routed	

2. В открывшемся окне выберите пункт «IP Management» → «DHCP» → «Activate»:



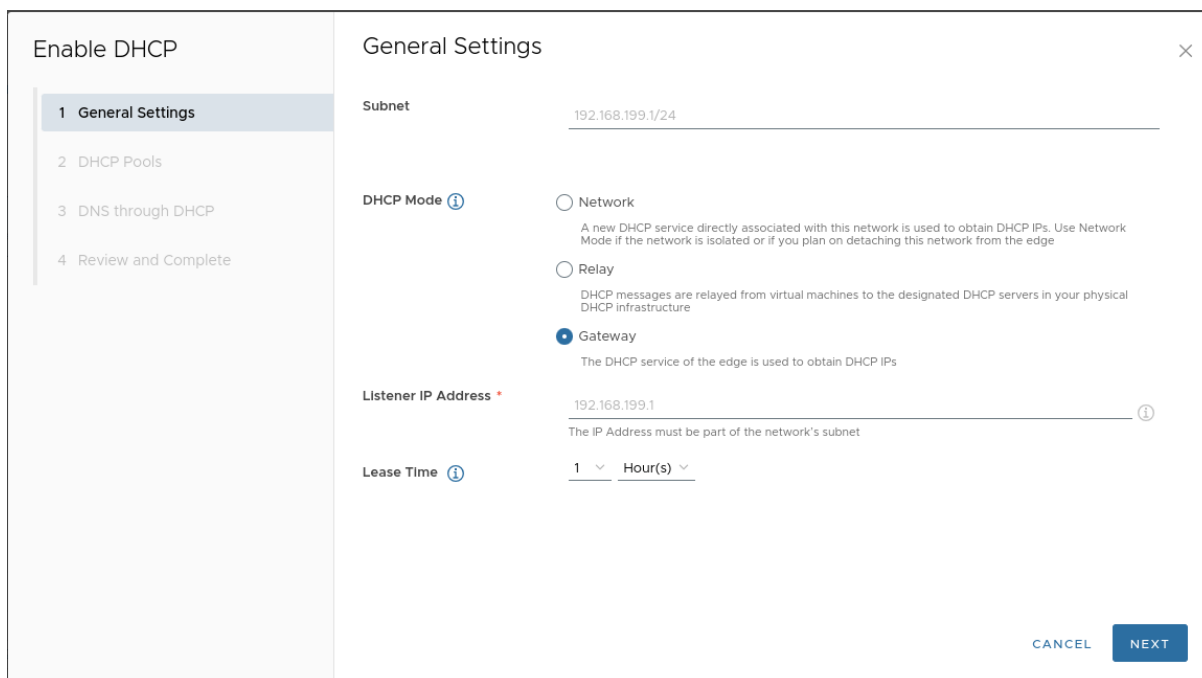
General
IP Management
Static IP Pools
DHCP
IP Usage
Security Groups

DHCP

ACTIVATE

i You haven't activated DHCP for this Network yet.
DHCP automates IP address assignment to Virtual Machines

3. На вкладке «General settings» настройте параметры аналогично примеру:



Enable DHCP

- 1 General Settings**
- 2 DHCP Pools
- 3 DNS through DHCP
- 4 Review and Complete

General Settings

Subnet: 192.168.199.1/24

DHCP Mode

☐ Network
A new DHCP service directly associated with this network is used to obtain DHCP IPs. Use Network Mode if the network is isolated or if you plan on detaching this network from the edge

☐ Relay
DHCP messages are relayed from virtual machines to the designated DHCP servers in your physical DHCP infrastructure

☒ **Gateway**
The DHCP service of the edge is used to obtain DHCP IPs

Listener IP Address: 192.168.199.1
The IP Address must be part of the network's subnet

Lease Time: 1 Hour(s)

CANCEL **NEXT**

4. Добавьте пул:

Enable DHCP

- General Settings
- DHCP Pools**
- DNS through DHCP
- Review and Complete

DHCP Pools

ADD

Pools
192.168.199.20-192.168.199.254

1 item(s)

Total IP addresses: 235

CANCEL
PREVIOUS
NEXT

5. Укажите адреса DNS-серверов:

Enable DHCP

- General Settings
- DHCP Pools
- DNS through DHCP**
- Review and Complete

DNS through DHCP

When a VM is connected to a network using DHCP mode, it will obtain these DNS IPs from the DHCP service. You can add up to two DNS servers using IPv4 or IPv6 addresses for both.

DNS Server 1
8.8.8.8

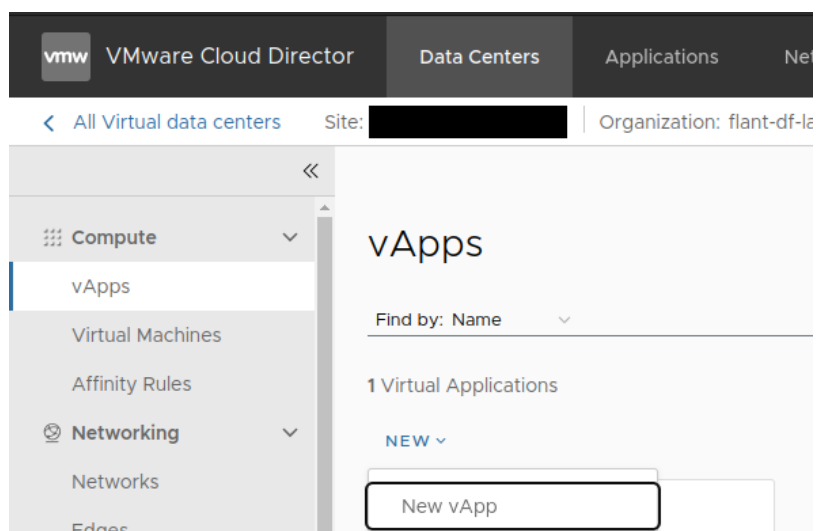
DNS Server 2
1.1.1.1

CANCEL
PREVIOUS
NEXT

Добавление vApp

i Инструкция актуальна только для схемы размещения **Standard**.

1. Перейдите на вкладку «Data Centers» → «vApps» → «NEW» → «New vApp»:



2. Укажите имя и включите vApp:

New vApp

Name *

deckhouse

Description

Power on

☒

Virtual Machines	OS	Compute

ADD VIRTUAL MACHINE

CANCEL

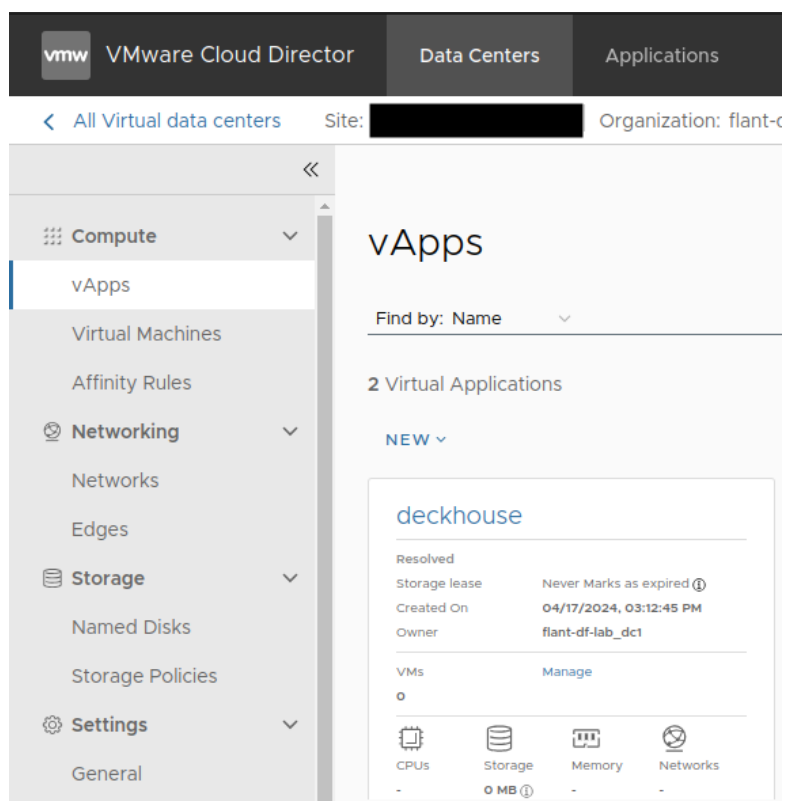
CREATE

Добавление сети к vApp

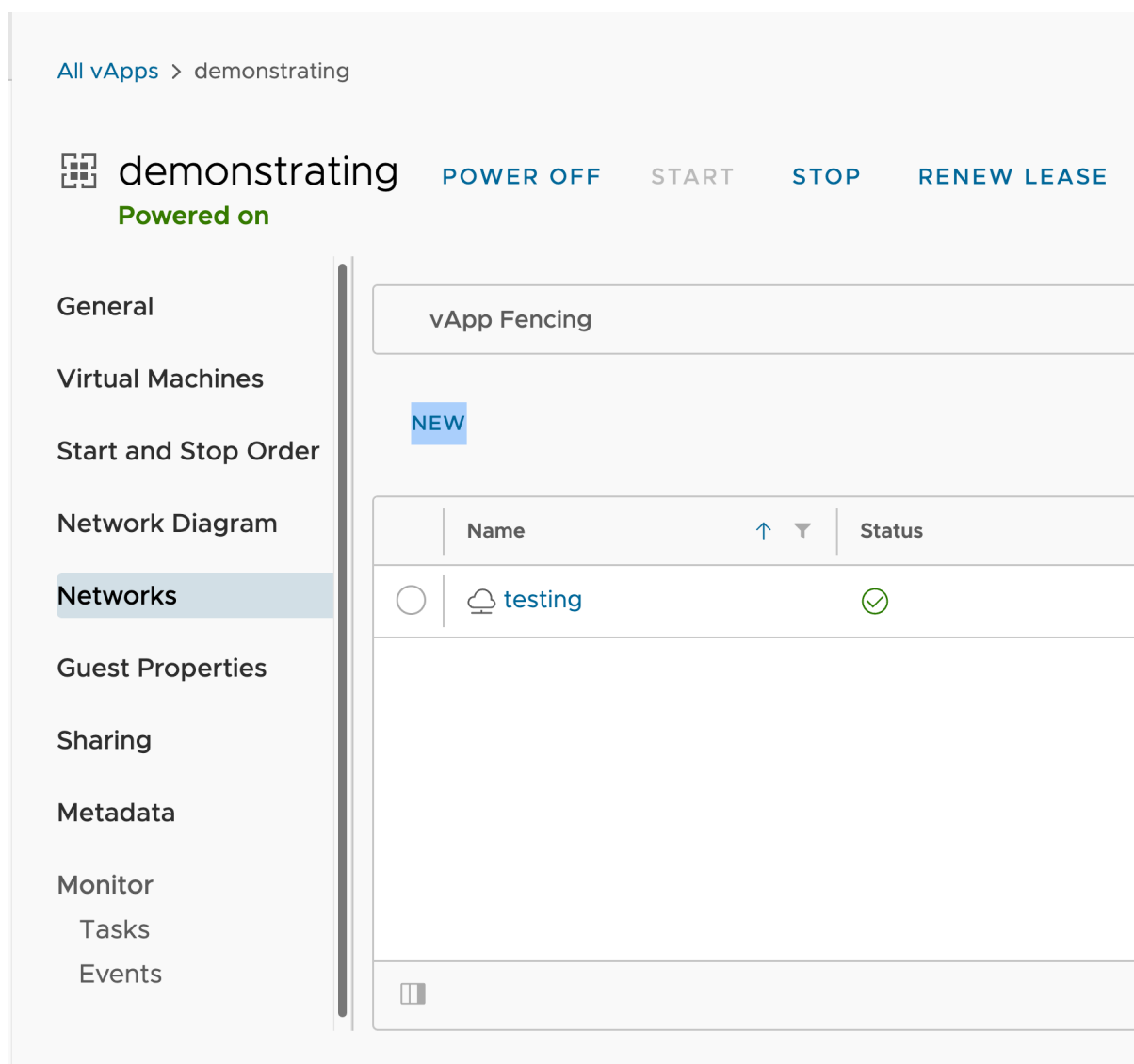
i Инструкция актуальна только для схемы размещения **Standard**.

После создания vApp присоедините к ней созданную внутреннюю сеть.

1. Перейдите на вкладку «Data Centers» → «vApps» и откройте необходимый vApp:



2. Перейдите на вкладку «Networks» и нажмите «NEW»:



3. В появившемся окне выберите тип «Direct» и выберите сеть:

Add Network to deckhouse
×

Type
☒ Direct ☐ Isolated

Org VDC Network Connection *

	Name ↑	Status	Organization VDC	Gateway CIDR	Network Type	Connected To	IP Pool Consumed	Shared	Route Advertised
☒	internal	✔	flant-df-la...	192.168.199.1/24	Routed	-	-		-

1 - 1 of 1 network(s)

CANCEL ADD

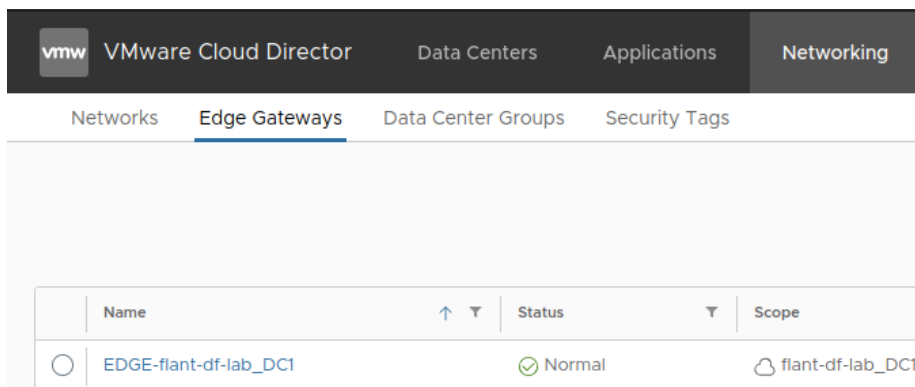
Входящий трафик

Входящий трафик необходимо направить на edge router (порты 80, 443) при помощи правил DNAT на выделенный адрес во внутренней сети. Этот адрес поднимается при помощи MetalLB в L2-режиме на выделенных frontend-узлах.

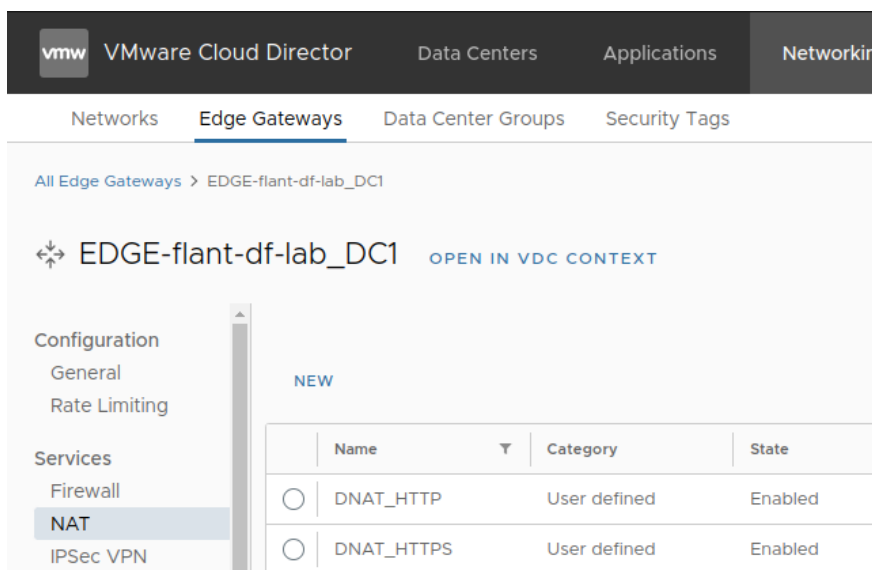
Настройка правил DNAT/SNAT на edge gateway

i Инструкция актуальна только для схемы размещения Standard.

1. Перейдите на вкладку «Networking» → «Edge Gateways», откройте edge gateway:



2. Перейдите на вкладку «Services» → «NAT»:



3. Добавьте следующие правила:

NAT										
Rules Logs										
NEW										
	Name	Category	State	NAT Action	External IP	Internal IP	Application	External Port	Destination IP	
☐	NAT-frontend-http	User de...	Active	DNAT		192.168.199.10	HTTP	80	-	
☐	NAT-frontend-https	User de...	Active	DNAT		192.168.199.10	HTTPS	443	-	
☐	NAT-master-0	User de...	Active	DNAT		192.168.199.2	SSH	22	-	

Первые два правила используются для входящего трафика, а третье — для доступа по SSH к узлу с control plane (без этого правила установка будет невозможна).

4. Чтобы виртуальные машины могли выходить в интернет, настройте правила SNAT, следуя примеру:

NAT										
Rules Logs										
NEW										
	Name	Category	State	NAT Action	External IP	Internal IP	Application	External Port	Destination IP	Logging
☐	SNAT	User de...	Active	SNAT		192.168.199.0/24	-	Any	-	No

Данное правило позволит виртуальным машинам из подсети 192.168.199.0/24 выходить в интернет.

Выпуск сертификатов Let's Encrypt

i Инструкция актуальна только для схемы размещения Standard.

При использовании cert-manager и ACME-проверки типа HTTP-01 может потребоваться настройка Hairpin NAT (NAT loopback) на Edge Gateway.

Это необходимо в сценарии, когда доменное имя Ingress резолвится во внешний IP-адрес Edge Gateway, а запрос к этому адресу выполняется **изнутри** кластера (из сети узлов). В таком случае трафик должен быть корректно возвращён обратно во внутреннюю сеть на адрес Ingress (например, адрес MetalLB).

Это важно для выпуска сертификатов Let's Encrypt, так как cert-manager выполняет предварительную проверку доступности challenge URL до обращения к ACME-провайдеру. Если URL недоступен изнутри кластера, выпуск сертификата не начнётся, даже если снаружи адрес открывается корректно.

Пример (сеть кластера 192.168.199.0/24):

- внутренняя сеть узлов: 192.168.199.0/24;
- внешний IP-адрес Edge Gateway: 194.117.83.19;
- внутренний адрес Ingress (например, MetalLB): 192.168.199.251.

В этом случае необходимо настроить Hairpin NAT для трафика из сети 192.168.199.0/24, направленного на внешний IP 194.117.83.19, с трансляцией на внутренний адрес Ingress 192.168.199.251.

i Для Edge Gateway на базе NSX-V настройка Hairpin NAT может быть обязательной.

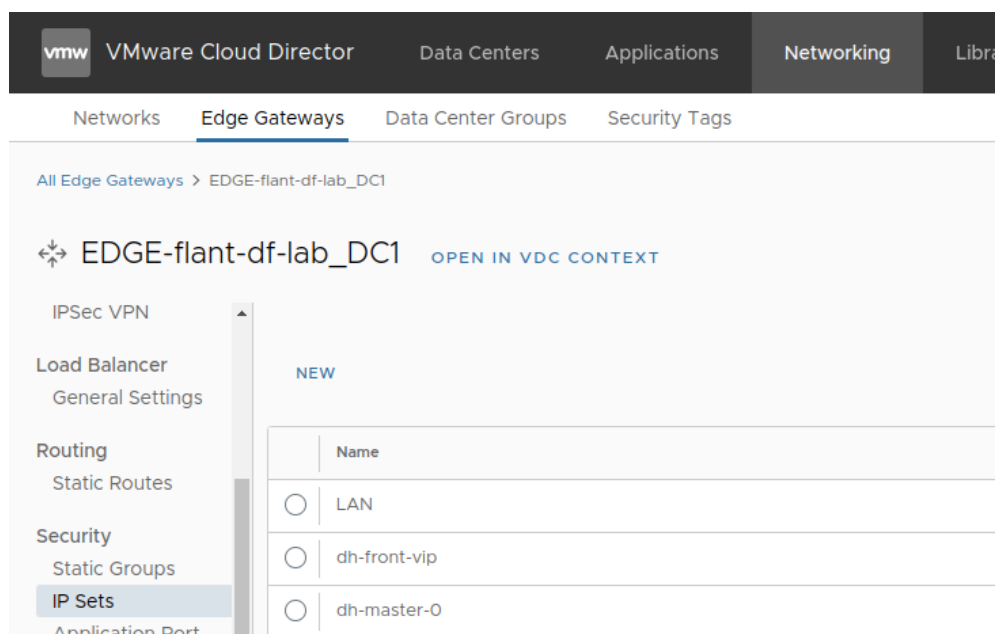
Для NSX-T loopback-сценарий часто поддерживается по умолчанию, но фактическое поведение зависит от конфигурации провайдера VMware Cloud Director.

Настройка firewall

i Инструкция актуальна только для схемы размещения Standard.

После настройки DNAT настройте firewall. Сначала необходимо настроить наборы IP-адресов (IP sets).

1. Перейдите на вкладку «Security» → «IP Sets»:



2. Создайте следующий набор IP (предполагается, что адрес MetalLB будет `.10` а адрес узла с control plane — `.2`):

Edit IP Set

Name *

LAN

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR ⓘ

192.168.199.0/24

ADD

MODIFY

REMOVE

↶ UNDO

DISCARD

SAVE

Edit IP Set

Name *

dh-front-vip

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR

192.168.199.10

ADD

MODIFY

REMOVE

UNDO

DISCARD

SAVE

Edit IP Set

Name *

dh-master-0

Description

IP Addresses

Enter an IPv4 or IPv6 address, range or CIDR

192.168.199.2

ADD

MODIFY

REMOVE

UNDO

DISCARD

SAVE

3. Добавьте следующие правила firewall:

Configuration

General

Rate Limiting

Services

Firewall

NAT

IPSec VPN

Load Balancer

General Settings

Routing

Static Routes

EDIT RULES

EDIT

DELETE

	#	Name	Category	State	Applications	Source	Destination	Action
☐	1	SNAT_LAN	User defined	Enabled	-	LAN	Any	Allow
☐	2	ACCESS_HTTP/HTTPS	User defined	Enabled	HTTPS, HTTP	Any	dh-front-vip	Allow
☐	3	ACCESS_SSH	User defined	Enabled	SSH	Any	dh-master-0	Allow
☐	4	Icmp	User defined	Enabled	ICMP ALL	Any	Any	Allow
☒		default_rule	Default	Enabled	-	Any	Any	Drop

Шаблон виртуальной машины

Работоспособность провайдера подтверждена только для шаблонов виртуальных машин на базе Ubuntu 22.04.

⚠ Отключите vApp/Guest OS Customization (Guest Customization, vApp Customization и аналогичные механизмы) для шаблона и виртуальных машин кластера в VMware Cloud Director. DKP выполняет первичную настройку узлов через `cloud-init` (datasource OVF/VMware GuestInfo). Включенная customization может конфликтовать с `cloud-init` и приводить к некорректной инициализации узла.

⚠ Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.

⚠ На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

В примере используется OVA-файл, предоставляемый Ubuntu, с двумя исправлениями. Исправления необходимы для корректного заказа CloudPermanent-узлов и для возможности подключать диски, созданные CSI.

Подготовка шаблона из OVA-файла

1. Скачайте OVA-файл:

	jammy-server-cloudimg-amd64.manifest	2024-02-21 13:01	18K	Package manifest file
	jammy-server-cloudimg-amd64.ova	2024-02-21 13:01	600M	VMware/Virtualbox OVA
	jammy-server-cloudimg-amd64.squashfs	2024-02-21 13:02	412M	Ubuntu Server 22.04 LTS (Jammy Jellyfish) dai

2. Перейдите на вкладку «Libraries» → «Catalogs» → «Каталог организации»:

VMware Cloud Director

Data Centers

Applications

Networking

Libraries

Administration

Monitor

More

flant-df-lab_dc1

Organization Administrator

All Catalogs > flant-df-lab_DC1

Content Libraries

vApp Templates

Organization VDC Templ...

Media & Other

Catalogs

Services

Service Library

flant-df-lab_DC1

SHARE

PUBLISH SETTINGS

SYNC

DELETE

ALL ACTIONS

General

vApp Templates

Media & Other

Metadata

NEW

	Name	Description	Status	Expired	Catalog Type	Catalog Publish Scope	Owner	Created On	Expires
	CentOS-Stream-9	https://cloud.centos.org/centos/9-stream/x86_64/...	Failed Creati...	No	Local	-	flant-df-lab_d...	11/30/2023, 07:32:27 PM	Never
	debian-11		Powered Off	No	Local	-	flant-df-lab_d...	02/21/2024, 02:11:44 PM	Never
	debian-11-cloud-template-2		Powered Off	No	Local	-	flant-df-lab_d...	02/21/2024, 09:19:19 AM	Never
	debian_ova		Powered Off	No	Local	-	flant-df-lab_d...	02/07/2024, 10:34:51 AM	Never
	focal-server-cloudimg-amd64	https://cloud-images.ubuntu.com/focal/20231129/f...	Failed Creati...	No	Local	-	flant-df-lab_d...	11/30/2023, 05:13:58 PM	Never
	nick-password		Powered Off	No	Local	-	flant-df-lab_d...	12/06/2023, 05:22:42 PM	Never

1 - 15 of 19 vApp Template(s)

3. Выберите скачанный шаблон и загрузите его в каталог:

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Select Source

Enter a URL from which to upload directly an OVF file.

URL

Browse to a location accessible from your computer, such as a local hard drive, a network share or a CD/DVD drive and select an OVF/OVA and all related files.

Browse

File(s):

- jammy-server-cloudimg-amd64 (1).ova

CANCEL

NEXT

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Review Details

Verify the OVF template details.

Product	Ubuntu 22.04 Server (20240221)
Version	-
Vendor	-
Download size	599.65 MB
Size on disks	0 B (thin provisioned) 10 GB (thick provisioned)
Description	-

CANCEL

PREVIOUS

NEXT

Страница 2002

Create vApp template from OVF

1 Select Source

2 Review Details

3 Select vApp Template Name

4 Ready to Complete

Select vApp Template Name

A vApp template is a virtual machine image that is loaded with an operating system, applications, and data. These templates ensure that virtual machines are consistently configured across an entire organization.

Name *

ubuntu-22.04-20240221

Description

Ubuntu 22.04 Server (20240221)

<https://cloud-images.ubuntu.com/jammy/20240221/jammy-server-cloudimg-amd64.ova>

Catalog *

flant-df-lab_DC1

CANCEL

PREVIOUS

NEXT

4. Создайте виртуальную машину из шаблона:

New VM

Name *

ubuntu-22.04-template

Computer Name *

ubuntu2204temp

Description

Type

☐ New

☒ From Template

Power on

☒

Templates

All VM templates

	Name	vApp Name	Catalog	OS	Compute	Storage
☐	ubuntu-jammy-22.04-cloudi...	ubuntu-22.04-20240221	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 4 Memory 8 GB	Policy
☒	ubuntu-jammy-22.04-cloudi...	ubuntu-22.04-20240221	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 2 Memory 1 GB	Policy
☐	ubuntu-jammy-22.04-cloudi...	Ubuntu 22.04 Server (2023...	flant-df-lab_...	Ubuntu Linux (64-bit)	CPU 2 Memory 1 GB	Policy

CANCEL

OK

New VM

Storage Policy

SSD-dc1-pub1-cl1 (VDC Default)

Compute

Sizing Policy

payg-4-8

Virtual CPUs

4

Cores per socket

1

Number of sockets

4

Memory

8 GB

NICs

Primary NIC	NIC	Connected	Network Adapter Type	Network	IP Mode	IP Address	MAC Address	
+	0		VMXNET3	LOCAL	DHCP		00:50:56:0c:17	

CANCEL

OK

⚠ Укажите пароль по умолчанию и публичный ключ. Они понадобятся, чтобы войти в консоль виртуальной машины.

New VM

Custom Properties

A Unique Instance ID for this instance

id-ovf

Specifies the instance id. This is required and used to determine if the machine should take "first boot" actions

Uri to seed instance data from

This field is optional, but indicates that the instance should "seed" user-data and meta-data from the given uri. If set to 'http://tinyurl.com/sm-' is given, meta-data will be pulled from http://tinyurl.com/sm-meta-data and user-data from http://tinyurl.com/sm-user-data. Leave this empty if you do not want to seed from a url.

ssh public keys

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDTXjTmx3hq2EPDQHWSJN7By1VNFZ8coll5tEeZDBVYae9Oxq4FZsKCb1e

This field is optional, but indicates that the instance should populate the default user's 'authorized_keys' with this value

Encoded user-data

In order to fit into a xml attribute, this value is base64 encoded . It will be decoded, and then processed normally as user-data.

Default User's password

123456

If set, the default user's password will be set to this value to allow password based login. The password will be good for only a single login. If set to the string 'RANDOM' then a random password will be generated, and written to the console.

End User License Agreements

There are no EULAs to review.

CANCEL

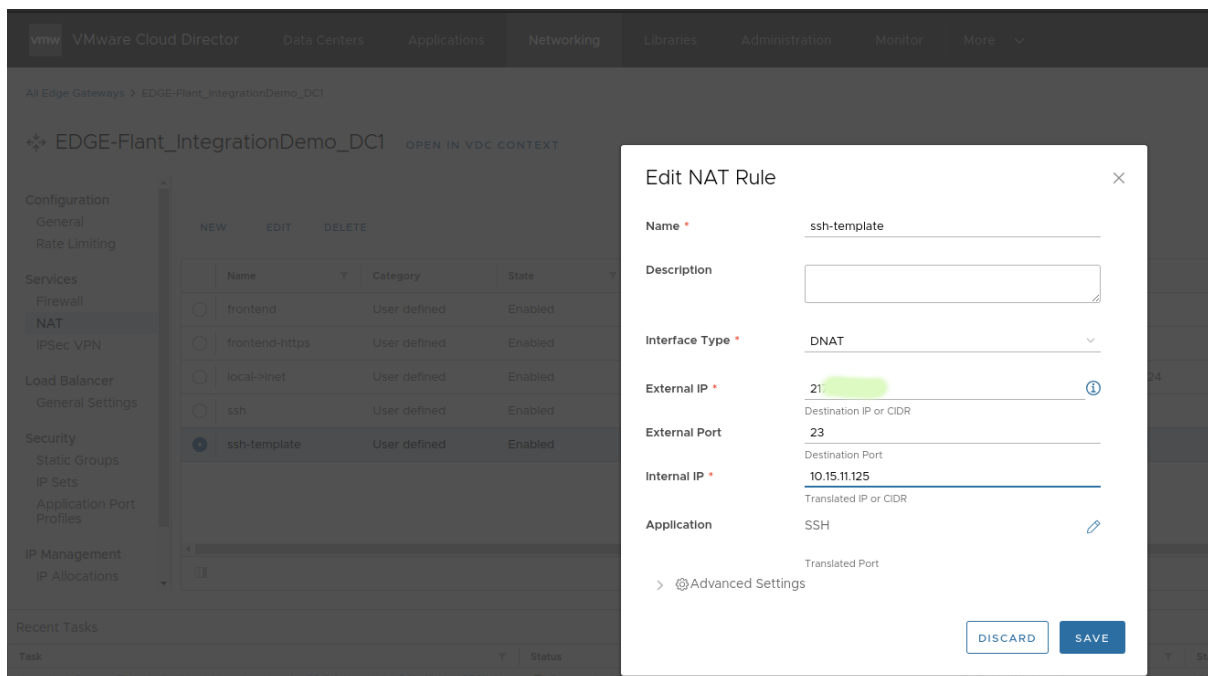
OK

Для подключения к виртуальной машине выполните следующие шаги:

1. Запустите виртуальную машину.

2. Дождитесь получение IP-адреса.

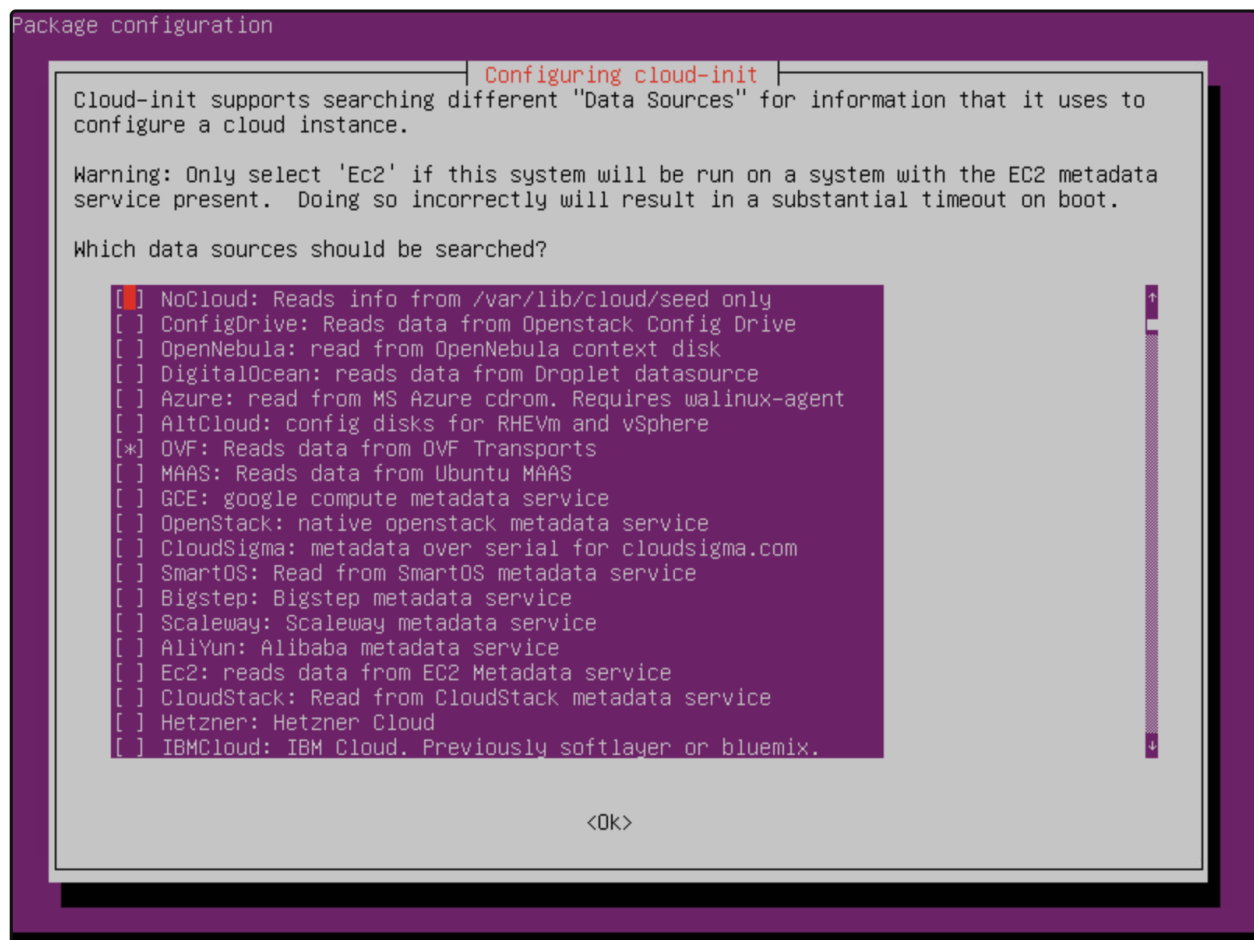
3. Пробросьте порт 22 до виртуальной машины:



Войдите на виртуальную машину по SSH и выполните следующие команды:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91_vmware_cust.cfg
dpkg-reconfigure cloud-init
```

В появившемся диалоговом окне оставьте галочку только у **OVF: Reads data from OVF transports**. Остальные пункты отключите:



Убедитесь, что в конфигурации `cloud-init` задан параметр `datasource_list`, с помощью следующей команды:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

Если вывод окажется пустым, выполните следующую команду:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

Выполните оставшиеся команды:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

Настройка шаблона в VCD

1. Выключите виртуальную машину и удалите все заполненные поля «Guest Properties»:

ubuntu-22.04
Powered off

POWER ONPOWER OFFLAUNCH WEB CONSOLELAUNCH REMOTE CONSOLEALL ACTIONS

General
Security Tags
Hardware
Removable Media
Hard Disks
Compute
NICs
Guest OS
Customization
Guest Properties
Metadata
Monitor
Tasks
Events

EDIT

A Unique Instance ID for this instance	Specifies the instance id. This is required and used to determine if the machine should take "first boot" actions
-	ubuntuguest Specifies the hostname for the appliance
Url to seed instance data from	This field is optional, but indicates that the instance should 'seed' user-data and meta-data from the given url. If set to 'http://tinyurl.com/sm-' is given, meta-data will be pulled from http://tinyurl.com/sm-meta-data and user-data from http://tinyurl.com/sm-user-data. Leave this empty if you do not want to seed from a url.
ssh public keys	ssh-rsa AAAAAB3NzaC1yc2EAAAADAQABAAQDTXJTmx3hq2EPDQHWSJN7By1VNFZ8coll5tEeZD BVYAe9Oxq4FZsKCb1aGlskDaiAHTxrbd2efoJTcPQLBSBM79dcELtqfKj9dtjy4S1W0mydvWb2o WLnvoaZX/H6pqjz8jrJAKXwXj2pWCOzXerwk9oSi4fCE7VbqsfT4bBfv27FN4/Vqa6iWiCc71oJo pL9DldtuIYDVUgOZOa+12J4hPCCSgEJK/r+ToHGboWxbCS/OAufXDw2W1vkVeaZUur5xwwAx lb3wM3WoS3BbwNIDYg9UB2D8+EZgNz1CCCPsyIEln7q8RnrTp0+H8V9LoWHSgh3VCWeW8 C/MnTW90IR This field is optional, but indicates that the instance should populate the default user's 'authorized_keys' with this value
Encoded user-data	In order to fit into a xml attribute, this value is base64 encoded . It will be decoded, and then processed normally as user-data

Edit Guest Properties for ubuntu-22.04

A Unique Instance ID for this instance

Url to seed instance data from

ssh public keys

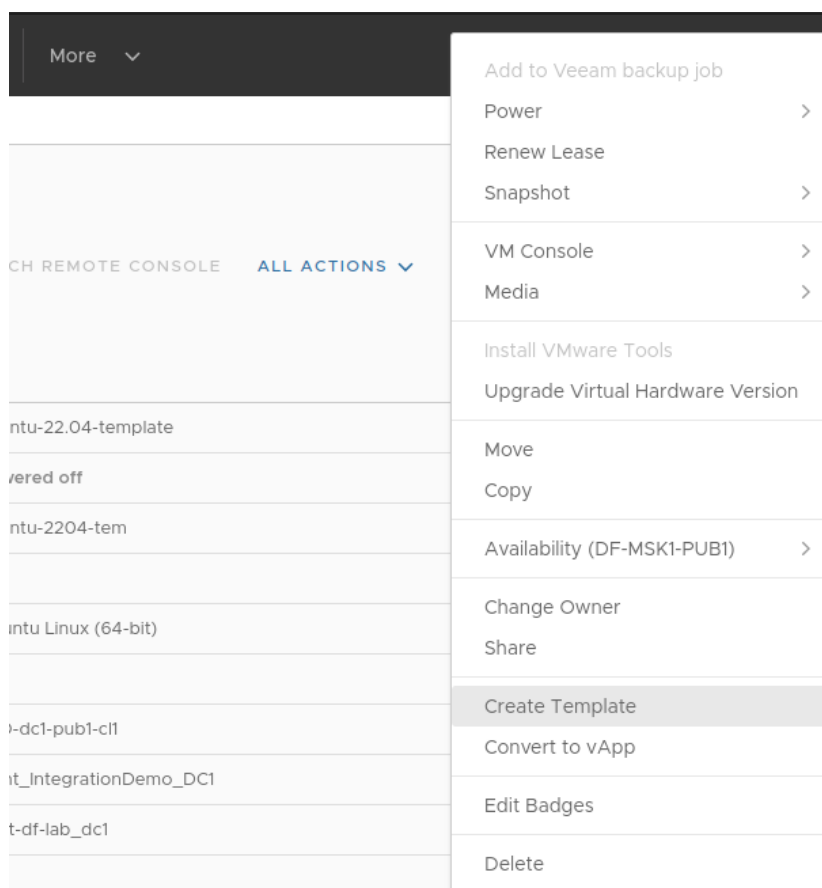
Encoded user-data

Default User's password

CANCEL

OK

2. Создайте шаблон виртуальной машины:



Add to Catalog: ubuntu-22.04-template

Add this vApp to catalog:

Catalog: * flant-df-lab_DC1

This catalog is local to your organization.

Overwrite catalog item: ☐

Name * ubuntu-22.04-template

Description

When using this template:

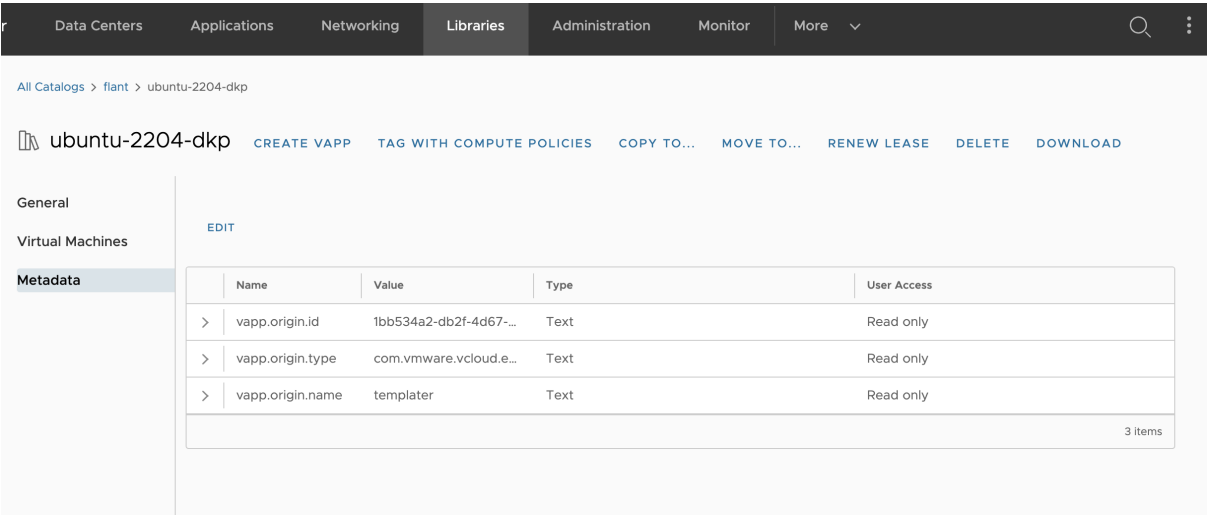
☐ Make identical copy ☒ Customize VM settings

This setting applies when creating a vApp based on this template. It is ignored when building a vApp using individual VMs from this template.

CANCEL **OK**

3. В созданном шаблоне перейдите на вкладку «Metadata» и добавьте шесть полей:

- guestinfo.metadata;
- guestinfo.metadata.encoding;
- guestinfo.userdata;
- guestinfo.userdata.encoding;
- disk.enableUUID;
- guestinfo.hostname.

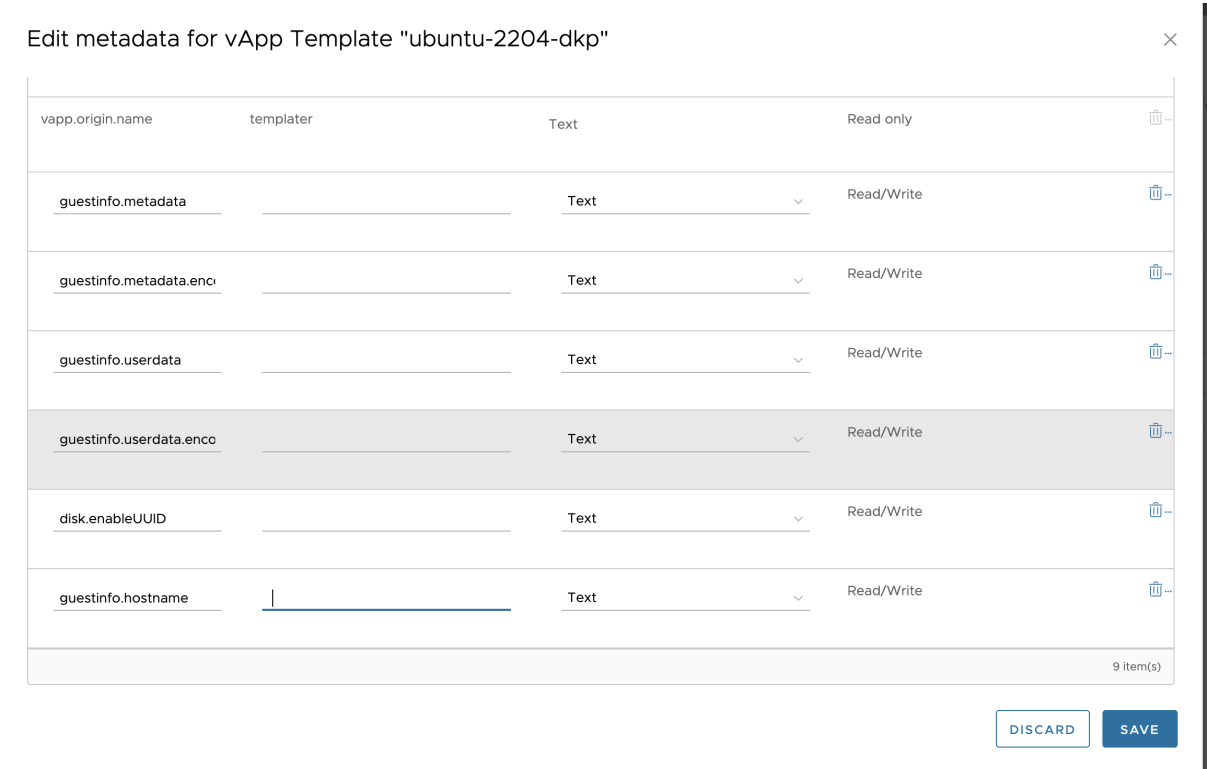


Для **каждого** поля в форме добавления/редактирования укажите:

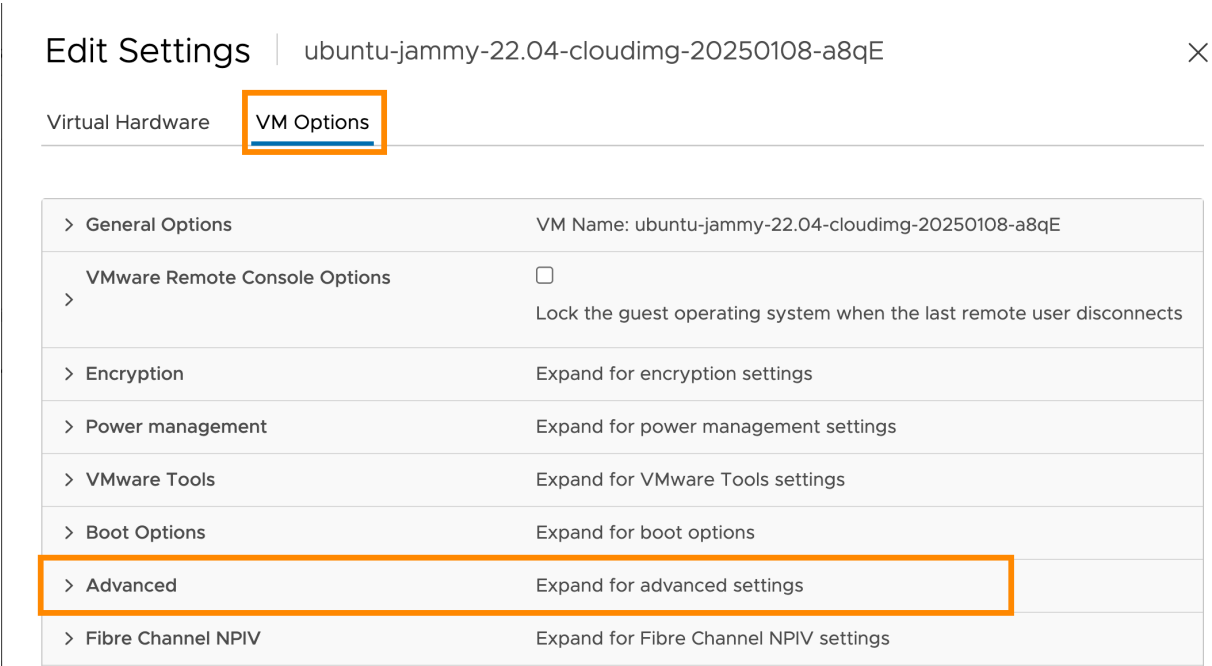
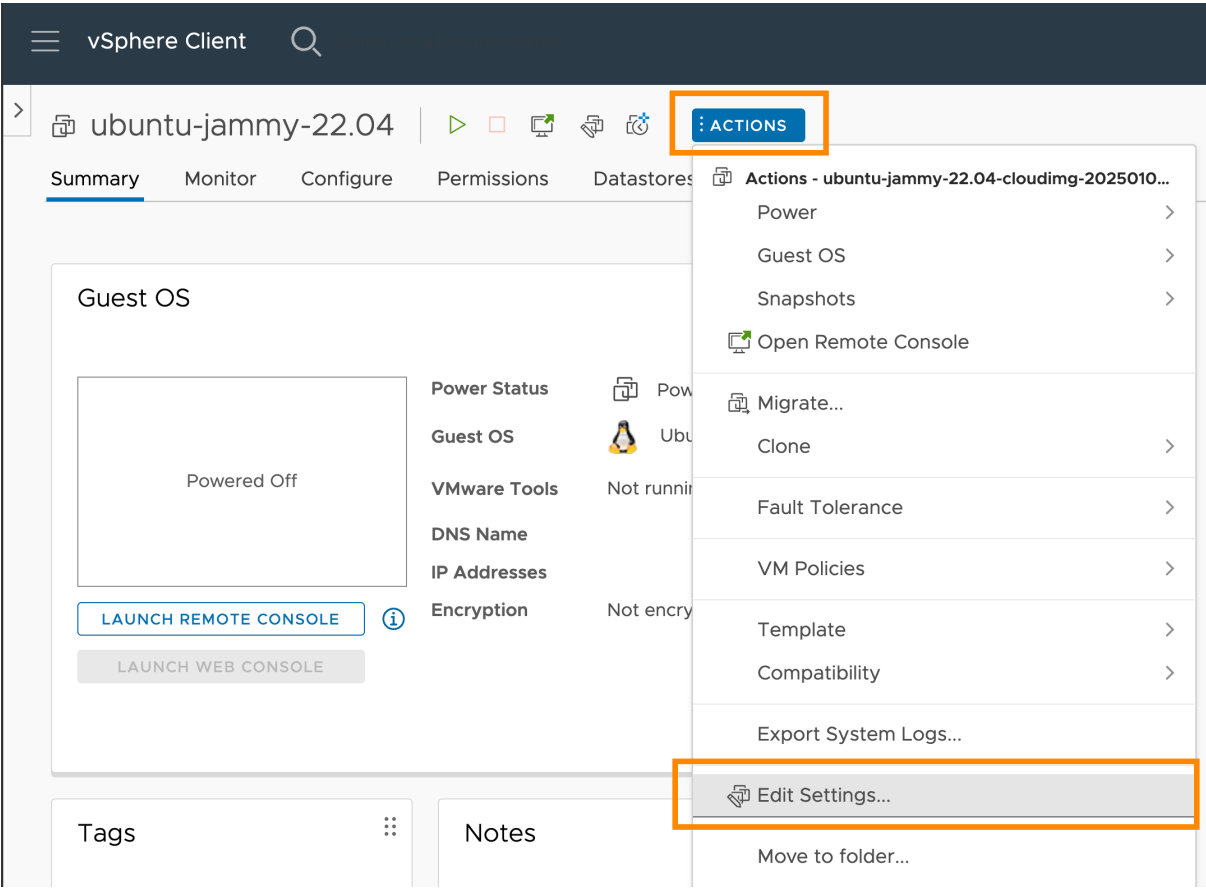
- «Type»: Text (текстовое значение);
- «User access»: Read/Write;
- «Value»: один пробел (space).

i

Интерфейс VCD может не сохранять метаданные с пустым значением. Пробел используется как техническое заполнение и не влияет на работу. Фактические значения будут подставлены автоматически при создании виртуальных машин.



4. В панели управления vCenter для шаблона включите параметр `disk.EnableUUID`:



Edit Settings | ubuntu-jammy-22.04-cloudimg-20250108-a8qE

Virtual Hardware VM Options

> General Options

VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE

VMware Remote Console Options

☐

Lock the guest operating system when the last remote user disconnects

> Encryption

Expand for encryption settings

> Power management

Expand for power management settings

> VMware Tools

Expand for VMware Tools settings

> Boot Options

Expand for boot options

< Advanced

Settings

☐ Disable acceleration

☒ Enable logging

Debugging and statistics

Run normally

Swap file location

☒ Default

Use the settings of the cluster or host containing the virtual machine.

☐ Virtual machine directory

Store the swap files in the same directory as the virtual machine.

☐ Datastore specified by host

Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.

Configuration Parameters

EDIT CONFIGURATION...

Configuration Parameters

⚠ Modify or add configuration parameters as needed for experimental features or as instructed by technical support. Empty values will be removed (supported on ESXi 6.0 and later).

ADD CONFIGURATION PARAMS

Name	Value
nvram	ubuntu-jammy-22.04-clc
svga.present	TRUE
pciBridge0.present	TRUE
pciBridge1.present	TRUE

Configuration Parameters



Modify or add configuration parameters as needed for experimental features or as instructed by technical support. Empty values will be removed (supported on ESXi 6.0 and later).

ADD CONFIGURATION PARAMS

Add New Configuration Params

Name	Value
disk.EnableUUID	TRUE

Name	Value
nvram	ubuntu-jammy-22.04-clc
svga.present	TRUE
pciBridge0.present	TRUE
pciBridge4.present	TRUE
pciBridge4.virtualDev	pcieRootPort
pciBridge4.functions	8
pciBridge5.present	TRUE
pciBridge5.virtualDev	pcieRootPort
pciBridge5.functions	8
pciBridge6.present	TRUE
pciBridge6.virtualDev	pcieRootPort
pciBridge6.functions	8
pciBridge7.present	TRUE
pciBridge7.virtualDev	pcieRootPort
pciBridge7.functions	8

CANCEL

OK

Использование хранилища

- VCD поддерживает CSI. Диски создаются как VCD Independent Disks.
- Guest property `disk.EnableUUID` должно быть разрешено для используемых шаблонов виртуальных машин.
- Deckhouse Kubernetes Platform поддерживает изменение размера дисков с версии v1.59.1.

Использование балансировщика нагрузки

- Компоненты DKP поддерживают ресурс Service типа LoadBalancer при установке в VMware Cloud Director (VCD).
- В качестве балансировщика используется VMware NSX Advanced Load Balancer (ALB или Avi Networks).
- Поддержка доступна **только** при использовании платформы виртуализации сети NSX-T.
- Балансировка должна быть включена на Edge Gateway вашим провайдером VCD. Проверить, включена ли балансировка, можно в разделе Edge Gateway → Load Balancer → General Settings — параметр State должен быть в статусе Active.
- Если балансировщик был активирован после успешного создания кластера DKP, компоненты автоматически подхватят изменения в течение часа (дополнительных действий не требуется).
- Для каждого открытого порта создаётся связка Pool + Virtual Service.
- При наличии межсетевого экрана необходимо создать разрешающее правило для внешнего IP-адреса балансировщика и соответствующих портов.

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

VCDInstanceClass

Scope: Cluster
Version: v1

Параметры группы виртуальных машин VCD, используемых `cluster-api-controller-manager` (модулем cloud-provider-vcd).

- **spec**
объект

Обязательный параметр

- **spec.additionalMetadata**
объект

Дополнительные строковые пары ключ-значение, которые будут добавлены в раздел метаданных (Metadata) каждой виртуальной машины master-узлов. Это позволяет более эффективно организовывать и классифицировать виртуальные машины в вашей системе виртуализации.

Свойства из этого поля будут объединены с уже существующими метаданными, хранящимися в поле `metadata` в корне спецификации `VCDProviderClusterConfiguration`. В случае совпадения ключей значения из этого поля будут иметь приоритет.

i **Внимание.** Изменение поля `additionalMetadata` приведет к пересозданию `CloudEphemeral NodeGroups`, использующих данный `VCDInstanceClass`.

- **spec.affinityRule**
объект

Определяет правило размещения, применяемое только к виртуальным машинам `NodeGroup`, ссылающимся на этот класс инстансов внутри кластера. Такие правила определяют, как узлы располагаются друг относительно друга на уровне гипервизоров, что может повысить отказоустойчивость и контролировать распределение рабочих нагрузок. Если не указано, правило размещения применяться не будет.

- **spec.affinityRule.polarity**
строка

Обязательный параметр

Полярность правила размещения. Должно быть либо `Affinity`, либо `AntiAffinity`. `Affinity` означает, что узлы должны быть размещены на одном хосте, в то время как `AntiAffinity` означает, что они должны быть размещены на разных хостах.

Допустимые значения: `Affinity`, `AntiAffinity`

- **spec.affinityRule.required**
булевый

Указывает, является ли правило размещения обязательным (`true`) или предпочтительным (`false`). Если установлено в `true`, правило должно быть соблюдено; если в `false`, правило применяется, если это возможно, но не является строго обязательным.

По умолчанию: `false`

- **spec.mainNetwork**
строка

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Пример:

```
mainNetwork: internal
```

- **spec.placementPolicy**
строка

`PlacementPolicy`, которая будет использоваться на данном компьютере.

Если `PlacementPolicy` не указана, для создания узлов будет использоваться `PlacementPolicy` по умолчанию.

Пример:

```
placementPolicy: Hypervisor1
```

- **spec.rootDiskSizeGb**
целочисленный

Размер root-диска. Значение указывается в гигабайтах.

По умолчанию: `20`

Пример:

```
rootDiskSizeGb: 20
```

- **spec.sizingPolicy**

строка

SizingPolicy, которая будет использоваться на данном компьютере.

Если SizingPolicy не указана, для создания узлов будет использоваться SizingPolicy по умолчанию.

Пример:

```
sizingPolicy: 4Cpu8Memory
```

- **spec.storageProfile**

строка

Обязательный параметр

StorageProfile, который будет использоваться на данном компьютере.

Пример:

```
storageProfile: ssd
```

- **spec.template**

строка

Обязательный параметр

Путь к шаблону OVA, который будет использоваться. Состоит из организации (опционально), каталога, названия или идентификатора шаблона. Если используется общий каталог, имя организации указывать обязательно (оно может отличаться от значения `provider.organization`).

Если в нескольких каталогах с одинаковым названием есть шаблоны с одинаковым именем, следует указать идентификатор шаблона.

Идентификатор шаблона можно найти в URL-адресе при открытии шаблона в пользовательском интерфейсе vCD. Например, если URL-адрес `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, то идентификатор шаблона будет `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Примеры:

```
template: Linux/ubuntu2204-cloud-ova
```

```
template: MyOrg/Linux/ubuntu2204-cloud-ova
```

```
template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

```
template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

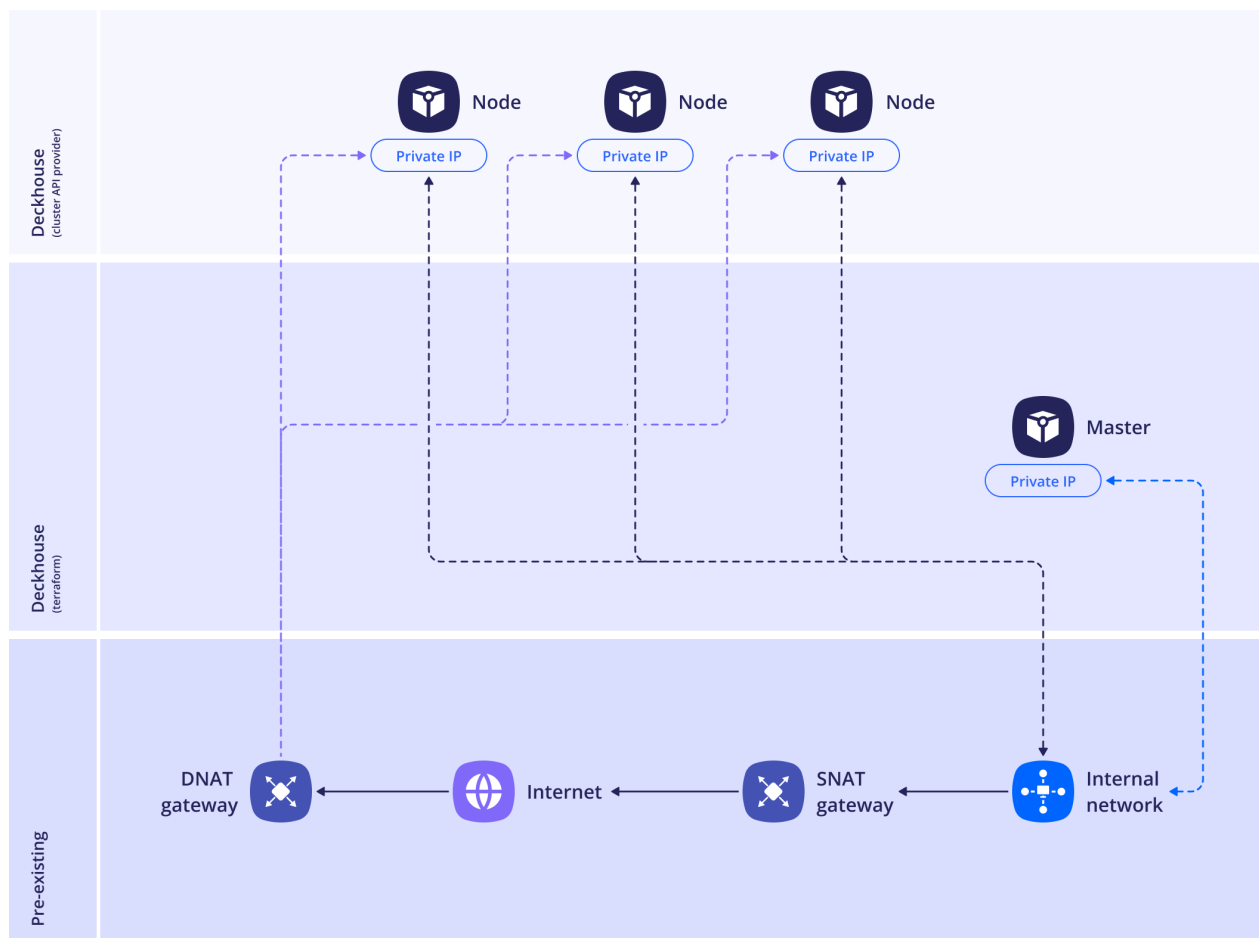
Примеры

Ниже представлен пример конфигурации VCDInstanceClass для эфемерных узлов cloud-провайдера VMware Cloud Director.

Пример конфигурации кастомного ресурса VCDInstanceClass

```
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova
```

Cloud provider - VMware Cloud Director: схемы размещения

Standard

Пример конфигурации схемы размещения:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
mainNetwork: internal
masterNodeGroup:
  replicas: 1
instanceClass:
  storageProfile: "Fast vHDD"
  sizingPolicy: 4cpu8mem
  template: "catalog/Ubuntu 22.04 Server"
mainNetworkIPAddresses:
  - 192.168.199.2
```

WithNAT



При использовании данной схемы размещения необходимо уточнить у администратора тип платформы сетевой виртуализации и указать его в параметре `edgeGateway.type`. Поддерживаются два варианта: `NSX-T` и `NSX-V`.

Для обеспечения административного доступа к узлам кластера разворачивается бастийон. Параметры для его настройки описываются в секции `bastion`.

Если Edge Gateway работает на базе `NSX-T`, в созданной сети для узлов автоматически активируется DHCP-сервер. Он будет выделять IP-адреса, начиная с 30-го адреса в подсети и до предпоследнего (перед broadcast-адресом). Начальный адрес DHCP-пула можно изменить с помощью параметра `internalNetworkDHCPPoolStartAddress`.

Если используется `NSX-V`, DHCP необходимо настроить вручную. В противном случае узлы, ожидающие получение IP-адреса по DHCP, не смогут его получить.



Не рекомендуется использовать динамическую адресацию для первого master-узла совместно с `NSX-V`.

Схема размещения предполагает автоматическое создание следующих правил NAT:

- SNAT — трансляция адресов внутренней сети узлов во внешний адрес, указанный в параметре `edgeGateway.externalIP`.
- DNAT — трансляция внешнего адреса и порта, заданных в параметрах `edgeGateway.externalIP` и `edgeGateway.externalPort`, на внутренний IP-адрес бастийона по порту 22 (протокол TCP) для обеспечения административного доступа по SSH.



Если Edge Gateway обеспечивается средствами `NSX-V`, то для построения правил необходимо указать имя и тип сети, к которым правило будет привязано в свойствах `edgeGateway.NSX-V.externalNetworkName` и `edgeGateway.NSX-V.externalNetworkType` соответственно. Как правило, это сеть, подключённая к Edge Gateway в разделе `Gateway Interfaces` и имеющая внешний IP-адрес.

Дополнительно возможно создание правил брандмауэра отдельным свойством `createDefaultFirewallRules`.



Если Edge Gateway обеспечивается средствами `NSX-T`, то существующие в Edge Gateway правила будут перезаписаны. Предполагается, что использование данной опции подразумевает размещение одного кластера на Edge Gateway.

Будут созданы следующие правила:

- Разрешение любого исходящего трафика;
- Разрешение входящего трафика по протоколу TCP и 22 порту для соединения с узлами кластера по SSH;
- Разрешение любого входящего трафика по протоколу ICMP;
- Разрешение входящего трафика по протоколам TCP и UDP и портам 30000-32767 для использования `NodePort`.

Пример конфигурации схемы размещения с использованием **NSX-T**:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
      - 192.168.199.2
```

Пример конфигурации схемы размещения с использованием **NSX-V**:

```

---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpulmem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
      - 192.168.199.2

```

Cloud provider — VMware vSphere

Модуль `cloud-provider-vsphere` обеспечивает интеграцию Deckhouse Kubernetes Platform с VMware vSphere. Он предоставляет возможность модулю `node-manager` использовать ресурсы vSphere при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-vsphere`:

- Управление ресурсами vSphere через `cloud-controller-manager`:
 - создаёт сетевые маршруты для сети PodNetwork на стороне vSphere;
 - актуализирует метаданные виртуальных машин и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в vSphere.
- Заказ дисков через CSI на datastore. По умолчанию используются CNS-тома с изменением размера на лету. Режим First-Class Disk (FCD) доступен как legacy и настраивается параметром `compatibilityFlag`.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-vsphere`.
- Заказ CloudEphemeral-узлов через Machine Controller Manager (MCM). Параметры виртуальных машин задаются в ресурсе VsphereInstanceClass.
- Регистрация в модуле `node-manager`, чтобы VsphereInstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.

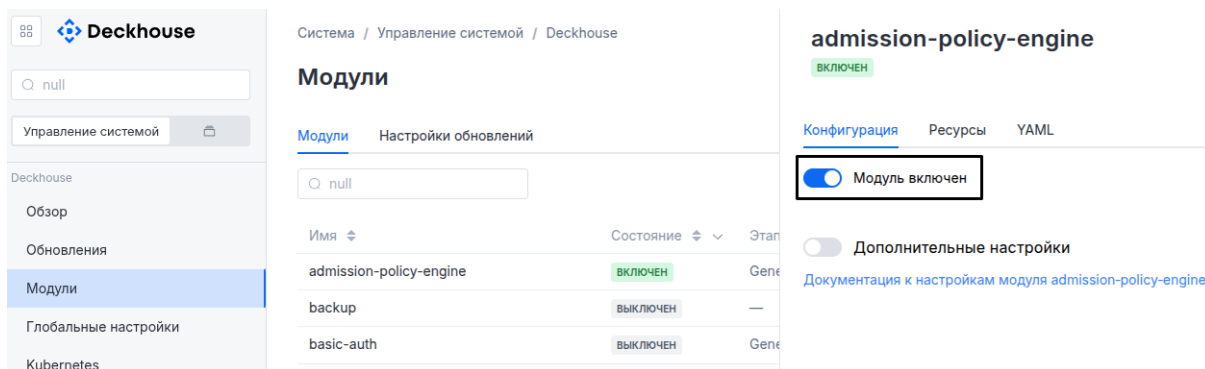
Настройки

Модуль автоматически включается для всех облачных кластеров, развернутых в vSphere.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vsphere`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-vsphere`:

```
d8 system module enable cloud-provider-vsphere
```

- **С помощью ModuleConfig cloud-provider-vsphere.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-vsphere` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  enabled: true
```

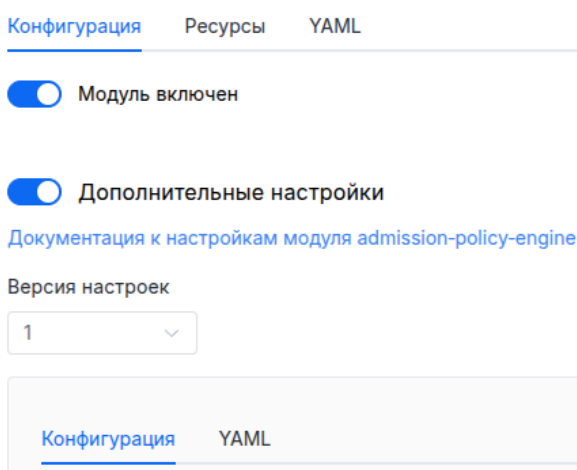
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vsphere`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `cloud-provider-vsphere` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-vsphere`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-vsphere` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-vsphere
```


Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-vsphere`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Если поле `.storageClass.compatibilityFlag` существует, перепишите значение с заглавной буквы.

Если control plane кластера размещен на виртуальных машинах или bare-metal-серверах, cloud-провайдер использует настройки модуля `cloud-provider-vsphere` в конфигурации Deckhouse (см. ниже). Иначе, если control plane кластера размещен в облаке, cloud-провайдер использует структуру VsphereClusterConfiguration для настройки.

Количество и параметры процесса заказа машин в облаке настраиваются в custom resource `NodeGroup` модуля `node-manager`, в котором также указывается название используемого для этой группы узлов инстанс-класса (параметр `cloudInstances.classReference NodeGroup`). Инстанс-класс для cloud-провайдера vSphere — это custom resource `VsphereInstanceClass`, в котором указываются конкретные параметры самих машин.

Storage

Модуль автоматически создает StorageClass для каждого Datastore и DatastoreCluster из зон (зоны).

Также он позволяет настроить имя StorageClass'a, который будет использоваться в кластере по умолчанию (параметр `default`) и отфильтровать ненужные StorageClass'ы (параметр `exclude`).

CSI

Подсистема хранения по умолчанию использует CNS-диски с возможностью изменения их размера на лету. Но также поддерживается работа и в legacy-режиме с использованием FCD-дисков. Поведение настраивается параметром `compatibilityFlag`.

Важная информация об увеличении размера PVC

Из-за особенностей работы volume-resizer CSI и vSphere API после увеличения размера PVC нужно сделать следующее:

1. На узле, где находится под, выполнить команду `d8 k cordon <имя_узла>`.
2. Удалить под.
3. Убедиться, что изменение размера прошло успешно. В объекте PVC *не будет* condition `Resizing`.

i Состояние `FileSystemResizePending` не является проблемой.

4. На узле, где находится под, выполнить команду `d8 k uncordon <имя_узла>`.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.disableTimesync**
булевый
Отключение синхронизации времени со стороны vSphere.

i **Внимание.** Отключение этого параметра не отключает NTP-демоны в гостевой ОС, а лишь отключает корректировки времени со стороны ESXi.

- **settings.externalNetworkNames**

массив строк

Имена сетей (не полный путь, а просто имя), подключенных к виртуальным машинам и используемых `vsphere-cloud-controller-manager` для предоставления ExternalIP в `.status.addresses` в Node API объект.

- **settings.host**

строка

Доменное имя сервера vCenter.

- **settings.insecure**

булевый

Установите `true`, если vCenter использует самоподписанный сертификат.

- **settings.internalNetworkNames**

массив строк

Имена сетей (не полный путь, а просто имя), подключенных к виртуальным машинам и используемых `vsphere-cloud-controller-manager` для предоставления InternalIP в `.status.addresses` в Node API объект.

- **settings.nsxt**

объект

Поддержка cloud controller manager'ом балансировщиков (LoadBalancer) в Vsphere через NSX-T.

- **settings.nsxt.defaultIpPoolName**

строка

Обязательный параметр

Имя пула IP-адресов по умолчанию, который используется, если в SVC не установлена аннотация `loadbalancer.vmware.io/class`.

Пример:

```
defaultIpPoolName: pool1
```

- **settings.nsxt.defaultTcpAppProfileName**

строка

Имя профиля по умолчанию, используемого для TCP-соединений в NSX-T.

По умолчанию: `default-tcp-lb-app-profile`

Примеры:

```
defaultTcpAppProfileName: default-tcp-lb-app-profile
```

```
defaultTcpAppProfileName: tcp-profile1
```

- **settings.nsxt.defaultUdpAppProfileName**

строка

Имя профиля по умолчанию, используемого для UDP-соединений в NSX-T.

По умолчанию: `default-udp-lb-app-profile`

Примеры:

```
defaultUdpAppProfileName: default-udp-lb-app-profile
```

```
defaultUdpAppProfileName: udp-profile1
```

- **settings.nsxt.host**

строка

Обязательный параметр

Адрес NSX-T.

Пример:

```
host: 1.2.3.4
```

- **settings.nsxt.insecureFlag**

булевый

Должен быть установлен в `true`, если NSX-T использует самоподписанный сертификат.

Примеры:

```
insecureFlag: true
```

```
insecureFlag: false
```

■ **settings.nsxt.loadBalancerClass**
массив

Дополнительная секция, определяющая классы балансировщика (LoadBalancer Class) (чтобы использовать класс, установите аннотацию `loadbalancer.vmware.io/class: <ИМЯ КЛАССА>` на SVC).

Примеры:

```
loadBalancerClass: []
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
  tcpAppProfileName: profile2
  udpAppProfileName: profile3
```

■ **settings.nsxt.loadBalancerClass.ipPoolName**

строка

Обязательный параметр

Имя пула IP-адресов.

■ **settings.nsxt.loadBalancerClass.name**

строка

Обязательный параметр

Имя класса балансировщика (LoadBalancer Class) для установки аннотации `loadbalancer.vmware.io/class: <ИМЯ КЛАССА>` в SVC.

■ **settings.nsxt.loadBalancerClass.tcpAppProfileName**

строка

Имя профиля по умолчанию, используемого для TCP-соединений.

По умолчанию: `defaultTcpAppProfileName`

■ **settings.nsxt.loadBalancerClass.udpAppProfileName**

строка

Имя профиля по умолчанию, используемого для UDP-соединений.

По умолчанию: `defaultUdpAppProfileName`

■ **settings.nsxt.password**

строка

Обязательный параметр

Пароль пользователя NSX-T.

Пример:

```
password: password
```

■ **settings.nsxt.size**

строка

Размер сервиса балансировщика (LoadBalancer).

По умолчанию: `MEDIUM`

Допустимые значения: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

Пример:

```
size: SMALL
```

■ **settings.nsxt.tier1GatewayPath**

строка

Обязательный параметр

Путь к политике NSX-T tier1 gateway.

Пример:

```
tier1GatewayPath: "/path/tier1"
```

- **settings.nsxt.user**

строка

Обязательный параметр

Имя пользователя NSX-T.

Пример:

```
user: user
```

- **settings.password**

строка

Пароль.

- **settings.region**

строка

Тег, прикрепленный к vSphere Datacenter, в котором будут происходить все операции: заказ виртуальных машин, размещение их дисков на datastore, подключение к сети.

- **settings.regionTagCategory**

строка

Имя категории тегов, используемых для идентификации региона (vSphere Datacenter).

- **settings.sshKeys**

массив строк

Список публичных SSH-ключей в формате plain-text.

- **settings.storageClass**

объект

- **settings.storageClass.compatibilityFlag**

строка

Флаг, разрешающий использование старой версии CSI:

- `Legacy` — используется старая версия драйвера — только диски FCD, без онлайн-изменения размера;
- `Migration` — в этом случае оба драйвера будут доступны в кластере одновременно. Этот режим используется для миграции со старого драйвера.

Допустимые значения: `Legacy`, `Migration`

- **settings.storageClass.default**

Параметр устарел

строка

Имя StorageClass'a, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- произвольный StorageClass в кластере, имеющий default-аннотацию;
- первый (по алфавиту) StorageClass из тех, что создаются модулем.

i

Параметр устарел. Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: fast-lun102-7d0bf578
```

- **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) StorageClass'ов, которые не будут созданы в кластере.

Пример:

```
exclude:
- ".*-lun101-.*"
- slow-lun103-1c280603
```

- **settings.username**

строка

Имя пользователя.

i **Внимание.** Всегда указывайте имя пользователя вместе с доменом, например: `username@domain.local`.
Уточнить домен можно в панели управления vCenter.

- **settings.vmFolderPath**

строка

Путь до VirtualMachine Folder, в котором будут создаваться клонированные виртуальные машины.

- **settings.zoneTagCategory**

строка

Имя категории тегов, использующихся для идентификации зоны (vSphere Cluster).

- **settings.zones**

массив строк

Глобальное ограничение набора зон, с которыми работает данный cloud-провайдер.

Настройки провайдера

i Если control plane кластера размещен на виртуальных машинах или bare-metal-серверах, cloud-провайдер использует настройки модуля `cloud-provider-vsphere` в конфигурации Deckhouse. Иначе, если control plane кластера размещен в облаке, cloud-провайдер использует структуру `VsphereClusterConfiguration` для настройки.

Дополнительная информация о Vsphere Cloud Load Balancers.

VsphereClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в vSphere.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами». Информация об уменьшении количества узлов и удалении узлов приведена в разделе «Интеграция со службами VMware vSphere».

Пример:

```

apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
layout: Standard
vmFolderPath: folder/prefix
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: region2
zones:
- region2-a
externalNetworkNames:
- net3-k8s
internalNetworkNames:
- K8S_3
internalNetworkCIDR: 172.16.2.0/24
baseResourcePool: kubernetes/cloud
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    mainNetwork: net3-k8s
    additionalNetworks:
    - K8S_3
    datastore: lun10
    rootDiskSize: 50
    runtimeOptions:
      nestedHardwareVirtualization: false
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    datastore: lun10
    mainNetwork: net3-k8s
provider:
  server: "<SERVER>"
  username: "<USERNAME@DOMAIN.LOCAL>"
  password: "<PASSWORD>"
  insecure: true

```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **baseResourcePool**

строка

Относительный (от vSphere Cluster) путь до существующего родительского `resourcePool` для всех создаваемых (в каждой зоне) `resourcePool`'ов.

- **disableTimesync**

булевый

Отключение синхронизации времени со стороны vSphere.

i

Внимание. Отключение этого параметра не отключает NTP-демоны в гостевой ОС, а лишь отключает корректировки времени со стороны ESXi.

По умолчанию: `true`

- **externalNetworkNames**

массив строк

Имена сетей (не полный путь, а просто имя), подключенных к виртуальным машинам и используемых `vsphere-cloud-controller-manager` для проставления ExternalIP в `.status.addresses` в Node API объект.

Пример:

```
externalNetworkNames:
- MAIN-1
- public
```

- **internalNetworkCIDR**

строка

Подсеть для master-узлов во внутренней сети.

Адреса выделяются с десятого адреса. Например, для подсети `192.168.199.0/24` будут использованы адреса начиная с `192.168.199.10`.

Будет использоваться при использовании `additionalNetworks` в `masterInstanceClass`.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkNames**

массив строк

Имена сетей (не полный путь, а просто имя), подключенных к виртуальным машинам и используемых `vsphere-cloud-controller-manager` для предоставления InternalIP в `.status.addresses` в Node API объект.

Пример:

```
internalNetworkNames:
- KUBE-3
- devops-internal
```

- **kind**

строка

Обязательный параметр

Допустимые значения: `VsphereClusterConfiguration`

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.

i Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое поля `VsphereInstanceClass`.

- **masterNodeGroup.instanceClass.additionalNetworks**

массив строк

Список путей до сетей, которые будут подключены к виртуальной машине.

Указывается относительно `vSphere datacenter`.

Пример:

```
additionalNetworks:
- DEVOPS_32
- DEVOPS_50
```

- **masterNodeGroup.instanceClass.datastore**

строка

Обязательный параметр

Путь до Datastore, на котором будут созданы клонированные виртуальные машины.

Указывается относительно vSphere datacenter.

Пример:

```
datastore: lun-1201
```

■ **masterNodeGroup.instanceClass.mainNetwork**

строка

Обязательный параметр

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Указывается относительно vSphere datacenter.

Пример:

```
mainNetwork: k8s-msk-178
```

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses**

массив объектов

Список статических IP-адресов (с CIDR-префиксом), назначаемых (по очереди) узлам в основной сети (параметр `mainNetwork`).

По умолчанию включен DHCP-клиент.

Пример:

```
mainNetworkIPAddresses:
- address: 10.1.14.20/24
  gateway: 10.1.14.254
  nameservers:
    addresses:
    - 8.8.8.8
    - 8.8.4.4
```

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses.address**

строка

Обязательный параметр

IP-адрес с CIDR префиксом.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses.gateway**

строка

Обязательный параметр

IP-адрес шлюза по умолчанию.

Должен находиться в подсети, указанной в параметре `address`.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers**

объект

Обязательный параметр

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.addresses**

массив строк

Список DNS-серверов.

Пример:

```
addresses:
- 8.8.8.8
- 8.8.4.4
```

■ **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

■ **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.search**

массив строк

Список доменов для поиска на указанных DNS-серверах.

Пример:

```
search:
- tech.lan
```

■ **masterNodeGroup.instanceClass.memory**
целочисленный

Обязательный параметр

Количество памяти (в мегабайтах), выделенной виртуальной машине.

Пример:

```
memory: 8192
```

■ **masterNodeGroup.instanceClass.numCPUs**
целочисленный

Обязательный параметр

Количество виртуальных процессорных ядер, выделяемых виртуальной машине.

Пример:

```
numCPUs: 2
```

■ **masterNodeGroup.instanceClass.resourcePool**
строка

Путь до Resource Pool, в котором будут созданы клонированные виртуальные машины.

Указывается относительно зоны (vSphere Cluster).

Пример:

```
resourcePool: rp-2012
```

■ **masterNodeGroup.instanceClass.rootDiskSize**
целочисленный

Размер (в гигабайтах) корневого диска в виртуальной машине.

Если в template указан диск меньшего размера, автоматически произойдет его расширение.

Пример:

```
rootDiskSize: 50
```

■ **masterNodeGroup.instanceClass.runtimeOptions**
объект

Дополнительные (опциональные) параметры виртуальных машин.

■ **masterNodeGroup.instanceClass.runtimeOptions.cpuLimit**
целочисленный

Верхний лимит (в MHz) потребляемой частоты процессоров для создаваемых виртуальных машин.

■ **masterNodeGroup.instanceClass.runtimeOptions.cpuReservation**
целочисленный

Величина зарезервированной для виртуальной машины частоты CPU (в MHz).

■ **masterNodeGroup.instanceClass.runtimeOptions.cpuShares**
целочисленный

Относительная величина CPU Shares для создаваемых виртуальных машин.

■ **masterNodeGroup.instanceClass.runtimeOptions.memoryLimit**
целочисленный

Верхний лимит (в мегабайтах) потребляемой памяти для создаваемых виртуальных машин.

■ **masterNodeGroup.instanceClass.runtimeOptions.memoryReservation**
целочисленный

Процент зарезервированной для виртуальной машины памяти в кластере. В процентах относительно `.spec.memory`.

Допустимые значения: $0 \leq X \leq 100$

■ **masterNodeGroup.instanceClass.runtimeOptions.memoryShares**

целочисленный

Относительная величина Memory Shares для создаваемых виртуальных машин.

Допустимые значения: `0 <= X <= 100`■ **masterNodeGroup.instanceClass.runtimeOptions.nestedHardwareVirtualization**

булевый

Включение Hardware Assisted Virtualization на созданных виртуальных машинах.

■ **masterNodeGroup.instanceClass.template**

строка

Обязательный параметр

Путь до VirtualMachine Template, который будет клонирован для создания новой виртуальной машины.

Пример:

```
template: dev/golden_image
```

○ **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`○ **masterNodeGroup.zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

● **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

○ **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей VsphereInstanceClass.

■ **nodeGroups.instanceClass.additionalNetworks**

массив строк

Список путей до сетей, которые будут подключены к виртуальной машине.

Указывается относительно vSphere datacenter.

Пример:

```
additionalNetworks:
- DEVOPS_32
- DEVOPS_50
```

■ **nodeGroups.instanceClass.datastore**

строка

Обязательный параметр

Путь до Datastore, на котором будут созданы клонированные виртуальные машины.

Указывается относительно vSphere datacenter.

Пример:

```
datastore: lun-1201
```

■ **nodeGroups.instanceClass.mainNetwork**

строка

Обязательный параметр

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Указывается относительно vSphere datacenter.

Пример:

```
mainNetwork: k8s-msk-178
```

■ **nodeGroups.instanceClass.mainNetworkIPAddresses**

массив объектов

Список статических IP-адресов (с CIDR-префиксом), назначаемых (по очереди) узлам в основной сети (параметр `mainNetwork`).

По умолчанию включен DHCP-клиент.

Пример:

```
mainNetworkIPAddresses:
- address: 10.1.14.20/24
  gateway: 10.1.14.254
  nameservers:
    addresses:
      - 8.8.8.8
      - 8.8.4.4
```

■ **nodeGroups.instanceClass.mainNetworkIPAddresses.address**

строка

Обязательный параметр

IP-адрес с CIDR префиксом.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

■ **nodeGroups.instanceClass.mainNetworkIPAddresses.gateway**

строка

Обязательный параметр

IP-адрес шлюза по умолчанию.

Должен находиться в подсети, указанной в параметре `address`.

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

■ **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers**

объект

Обязательный параметр

■ **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.addresses**

массив строк

Список DNS-серверов.

Пример:

```
addresses:
- 8.8.8.8
- 8.8.4.4
```

■ **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

■ **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.search**

массив строк

Список доменов для поиска на указанных DNS-серверах.

Пример:

```
search:
- tech.lan
```

■ **nodeGroups.instanceClass.memory**

целочисленный

Обязательный параметр

Количество памяти (в мегабайтах), выделенной виртуальной машине.

Пример:

```
memory: 8192
```

■ **nodeGroups.instanceClass.numCPUs**

целочисленный

Обязательный параметр

Количество виртуальных процессорных ядер, выделяемых виртуальной машине.

Пример:

```
numCPUs: 2
```

■ **nodeGroups.instanceClass.resourcePool**

строка

Путь до Resource Pool, в котором будут созданы клонированные виртуальные машины.

Указывается относительно зоны (vSphere Cluster).

Пример:

```
resourcePool: rp-2012
```

■ **nodeGroups.instanceClass.rootDiskSize**

целочисленный

Размер (в гигабайтах) корневого диска в виртуальной машине.

Если в template указан диск меньшего размера, автоматически произойдет его расширение.

Пример:

```
rootDiskSize: 50
```

■ **nodeGroups.instanceClass.runtimeOptions**

объект

Дополнительные (опциональные) параметры виртуальных машин.

■ **nodeGroups.instanceClass.runtimeOptions.cpuLimit**

целочисленный

Верхний лимит (в MHz) потребляемой частоты процессоров для создаваемых виртуальных машин.

■ **nodeGroups.instanceClass.runtimeOptions.cpuReservation**

целочисленный

Величина зарезервированной для виртуальной машины частоты CPU (в MHz).

■ **nodeGroups.instanceClass.runtimeOptions.cpuShares**

целочисленный

Относительная величина CPU Shares для создаваемых виртуальных машин.

■ **nodeGroups.instanceClass.runtimeOptions.memoryLimit**

целочисленный

Верхний лимит (в мегабайтах) потребляемой памяти для создаваемых виртуальных машин.

■ **nodeGroups.instanceClass.runtimeOptions.memoryReservation**

целочисленный

Процент зарезервированной для виртуальной машины памяти в кластере. В процентах относительно `.spec.memory`.

Допустимые значения: $0 \leq X \leq 100$

■ **nodeGroups.instanceClass.runtimeOptions.memoryShares**

целочисленный

Относительная величина Memory Shares для создаваемых виртуальных машин.

Допустимые значения: $0 \leq X \leq 100$

■ **nodeGroups.instanceClass.runtimeOptions.nestedHardwareVirtualization**

булевый

Включение Hardware Assisted Virtualization на созданных виртуальных машинах.

- **nodeGroups.instanceClass.template**

строка

Обязательный параметр

Путь до VirtualMachine Template, который будет скопирован для создания новой виртуальной машины.

Пример:

```
template: dev/golden_image
```

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **nodeGroups.zones**

массив строк

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

- **nsxt**

объект

Поддержка cloud controller manager'ом балансировщиков (LoadBalancer) в Vsphere через NSX-T.

- **nsxt.defaultIpPoolName**

строка

Обязательный параметр

Имя пула IP-адресов по умолчанию, который используется, если в SVC не установлена аннотация `loadbalancer.vmware.io/class`.

Пример:

```
defaultIpPoolName: pool1
```

- **nsxt.defaultTcpAppProfileName**

строка

Имя профиля по умолчанию, используемого для TCP-соединений в NSX-T.

По умолчанию: `default-tcp-lb-app-profile`

Примеры:

```
defaultTcpAppProfileName: default-tcp-lb-app-profile
```

```
defaultTcpAppProfileName: tcp-profile1
```

- **nsxt.defaultUdpAppProfileName**

строка

Имя профиля по умолчанию, используемого для UDP-соединений в NSX-T.

По умолчанию: `default-udp-lb-app-profile`

Примеры:

```
defaultUdpAppProfileName: default-udp-lb-app-profile
```

```
defaultUdpAppProfileName: udp-profile1
```

- **nsxt.host**

строка

Обязательный параметр

Адрес NSX-T.

Пример:

```
host: 1.2.3.4
```

- **nsxt.insecureFlag**

булевый

Должен быть установлен в `true`, если NSX-T использует самоподписанный сертификат.

Примеры:

```
insecureFlag: true
```

```
insecureFlag: false
```

- **nsxt.loadBalancerClass**

массив

Дополнительная секция, определяющая классы балансировщика (LoadBalancer Class) (чтобы использовать класс, установите аннотацию `loadbalancer.vmware.io/class: <ИМЯ КЛАССА>` на SVC).

Примеры:

```
loadBalancerClass: []
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
  tcpAppProfileName: profile2
  udpAppProfileName: profile3
```

■ **nsxt.loadBalancerClass.ipPoolName**

строка

Обязательный параметр

Имя пула IP-адресов.

■ **nsxt.loadBalancerClass.name**

строка

Обязательный параметр

Имя класса балансировщика (LoadBalancer Class) для установки аннотации `loadbalancer.vmware.io/class: <ИМЯ КЛАССА>` в SVC.

■ **nsxt.loadBalancerClass.tcpAppProfileName**

строка

Имя профиля по умолчанию, используемого для TCP-соединений.

По умолчанию: `defaultTcpAppProfileName`

■ **nsxt.loadBalancerClass.udpAppProfileName**

строка

Имя профиля по умолчанию, используемого для UDP-соединений.

По умолчанию: `defaultUdpAppProfileName`

○ **nsxt.password**

строка

Обязательный параметр

Пароль пользователя NSX-T.

Пример:

```
password: password
```

○ **nsxt.size**

строка

Размер сервиса балансировщика (LoadBalancer).

По умолчанию: `MEDIUM`

Допустимые значения: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

Пример:

```
size: SMALL
```

○ **nsxt.tier1GatewayPath**

строка

Обязательный параметр

Путь к политике NSX-T tier1 gateway.

Пример:

```
tier1GatewayPath: "/path/tier1"
```

○ **nsxt.user**

строка

Обязательный параметр

Имя пользователя NSX-T.

Пример:

```
user: user
```

- **provider**

объект

Обязательный параметр

Параметры подключения к vCenter.

- **provider.insecure**

булевый

Установите `true`, если vCenter использует самоподписанный сертификат.

По умолчанию: `false`

- **provider.password**

строка

Обязательный параметр

Пароль.

- **provider.server**

строка

Обязательный параметр

Хост или IP-адрес vCenter сервера.

- **provider.username**

строка

Обязательный параметр

Имя пользователя.

i

Внимание. Обязательно указывайте имя пользователя вместе с доменом, например: `username@domain.local`.

Уточнить домен можно в панели управления vCenter.

- **region**

строка

Обязательный параметр

Тег, прикрепленный к vSphere Datacenter, в котором будут происходить все операции: заказ виртуальных машин, размещение их дисков на datastore, подключение к network.

- **regionTagCategory**

строка

Обязательный параметр

Имя категории тегов, используемых для идентификации региона (vSphere Datacenter).

По умолчанию: `k8s-region`

- **sshPublicKey**

строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **storagePolicyID**

строка

ID политики хранилищ SPBM (Storage Policy Based Management).

- **useNestedResourcePool**

булевый

Создание вложенного пула (`true`) или использование основного пула (`false`).

По умолчанию: `true`

- **vmFolderExists**

булевый

Установите значение в `true`, если путь заданный в `vmFolderPath` существует. Установка более одного кластера в папку невозможна.

По умолчанию: `false`

- **vmFolderPath**

строка

Обязательный параметр

Путь до VirtualMachine Folder, в котором будут создаваться клонированные виртуальные машины.

Пример:

```
vmFolderPath: dev/test
```

- **zoneTagCategory**

строка

Обязательный параметр

Имя категории тегов, использующихся для идентификации зоны (vSphere Cluster).

По умолчанию: `k8s-zone`

- **zones**

массив строк

Обязательный параметр

Глобальное ограничение набора зон, с которыми работает данный cloud-провайдер.

Подготовка окружения

Требования к окружению

Для корректной работы Deckhouse Kubernetes Platform с VMware vSphere необходимы:

- Доступ к vCenter;
- Пользователь с необходимым набором привилегий;
- Созданные теги и категории тегов в vSphere;
- Сети с DHCP и доступом в Интернет;
- Доступные shared Datastore на всех используемых ESXi;
- Версия vSphere — 7.x или 8.x с поддержкой механизма Online volume expansion;
- vCenter, доступный изнутри кластера с master-узлов;
- Созданный Datacenter, в котором должны быть настроены следующие объекты:
 1. VirtualMachine template:
 - Образ виртуальной машины должен использовать `Virtual machines with hardware version 15 or later` — это необходимо для работы online resize.
 - В образе должны быть установлены пакеты `open-vm-tools`, `cloud-init` и `cloud-init-vmware-guestinfo` — при использовании версии `cloud-init` ниже 21.3.
 2. Network:
 - Сеть должна быть доступна на всех ESXi, на которых планируется создание виртуальных машин.
 3. Datastore (один или несколько):
 - Datastore должен быть подключен ко всем ESXi, на которых планируется создание виртуальных машин.
 - На Datastore должен быть назначен тег из категории, указанной в параметре `zoneTagCategory` (по умолчанию — `k8s-zone`). Этот тег определяет зону.
 - Все Cluster в пределах одной зоны должны иметь доступ ко всем Datastore с той же зоной.
 4. Cluster:
 - В Cluster должны быть добавлены все используемые ESXi.
 - На Cluster должен быть назначен тег из категории, указанной в параметре `zoneTagCategory` (по умолчанию — `k8s-zone`). Этот тег определяет зону.
 5. Folder для создаваемых виртуальных машин:
 - Параметр опционален.
 - По умолчанию используется корневой каталог виртуальных машин.
 6. Role:
 - Роль должна содержать необходимый набор привилегий.
 7. User:
 - Пользователю должна быть назначена роль, указанная в предыдущем пункте.
- На созданный Datacenter должен быть назначен тег из категории, указанной в параметре `regionTagCategory` (по умолчанию — `k8s-region`). Этот тег определяет регион.

Список необходимых ресурсов vSphere

- **User** с необходимым набором привилегий.
- **Network** с DHCP и доступом в интернет.
- **Datacenter** с соответствующим тегом `k8s-region`.
- **Cluster** с соответствующим тегом `k8s-zone`.
- **Datastore** в любом количестве с соответствующими тегами.
- **Template** — подготовленный образ виртуальной машины.

Список необходимых привилегий

i О том, как создать и назначить роль пользователю, читайте в разделах «Настройка через vSphere Client» и «Настройка через govc».

Детальный список привилегий, необходимых для работы Deckhouse Kubernetes Platform в vSphere:

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
—	— (назначаются по умолчанию при создании роли)	System.Anonymous System.Read System.View	Базовый доступ к объектам vSphere Inventory, необходимый для работы всех компонентов интеграции Deckhouse с vSphere.
Cns	Searchable	Cns.Searchable	Поиск и сопоставление объектов Container Native Storage при работе CSI-драйвера с томами Kubernetes.
Datastore	Allocate space, Browse datastore, Low level file operations	Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement	Выделение дисков при создании виртуальных машин и заказе PersistentVolumes в кластере.
Folder	Create folder, Delete folder, Move folder, Rename folder	Folder.Create Folder.Delete Folder.Move Folder.Rename	Группировка кластера Deckhouse Kubernetes Platform в одном Folder в vSphere Inventory.
Global	Global tag, System tag	Global.GlobalTag Global.SystemTag	Доступ к глобальным и системным тегам, используемым Deckhouse Kubernetes Platform при работе с объектами vSphere.
vSphere Tagging	Assign or Unassign vSphere Tag, Assign or Unassign vSphere Tag on Object, Create vSphere Tag, Create vSphere Tag Category, Delete vSphere Tag, Delete vSphere Tag Category, Edit vSphere Tag, Edit vSphere Tag Category, Modify UsedBy Field for Category, Modify UsedBy Field for Tag	InventoryService.Tagging.AttachTag InventoryService.Tagging.ObjectAttachable InventoryService.Tagging.CreateTag InventoryService.Tagging.CreateCategory InventoryService.Tagging.DeleteTag InventoryService.Tagging.DeleteCategory InventoryService.Tagging.EditTag InventoryService.Tagging.EditCategory InventoryService.Tagging.ModifyUsedByForCategory InventoryService.Tagging.ModifyUsedByForTag	Deckhouse Kubernetes Platform использует теги для определения доступных ему объектов Datacenter, Cluster и Datastore, а также для определения виртуальных машин, находящихся под его управлением.
Network	Assign network	Network.Assign	Подключение сетей и port group к виртуальным машинам кластера Deckhouse Kubernetes Platform.
Resource	Assign virtual machine to resource pool, Create resource pool, Modify resource pool, Remove resource pool, Rename resource pool	Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool	Размещение виртуальных машин кластера Deckhouse Kubernetes Platform в целевом пуле ресурсов и управление этим пулом.
VM Storage Policies (Profile-driven Storage Privileges в vSphere 7)	View VM storage policies (Profile-driven storage view в vSphere 7)	StorageProfile.View	Просмотр политик хранения, используемых при создании виртуальных машин и динамическом заказе томов в кластере.

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
vApp	Add virtual machine, Assign resource pool, Create, Delete, Import, Power Off, Power On, View OVF Environment, vApp application configuration, vApp instance configuration, vApp resource configuration	VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig	Управление операциями, связанными с развертыванием и конфигурацией vApp и OVF-шаблонов, используемых при создании виртуальных машин.
Virtual Machine > Change Configuration	Add existing disk, Add new disk, Add or remove device, Advanced configuration, Set annotation, Change CPU count, Toggle disk change tracking, Extend virtual disk, Acquire disk lease, Modify device settings, Configure managedBy, Change Memory, Query unowned files, Configure Raw device, Reload from path, Remove disk, Rename, Reset guest information, Change resource, Change Settings, Change Swapfile placement, Upgrade virtual machine compatibility	VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement VirtualMachine.Config.UpgradeVirtualHardware	Управление жизненным циклом виртуальных машин кластера Deckhouse Kubernetes Platform.
Virtual Machine > Edit Inventory	Create new, Create from existing, Remove, Move	VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete VirtualMachine.Inventory.Move	Создание, удаление и перемещение виртуальных машин кластера Deckhouse Kubernetes Platform в инвентаре vSphere.
Virtual Machine > Guest Operations	Guest Operation Queries	VirtualMachine.GuestOperations.Query	Получение информации из гостевой операционной системы виртуальных машин.

Категория привилегий в UI	Список привилегий в UI	Список привилегий в API	Назначение привилегий в Deckhouse
Virtual Machine > Interaction	Answer question, Device connection, Guest operating system management by VIX API, Power Off, Power On, Reset, Configure CD media, Install VMware Tools	VirtualMachine.Interact. t.AnswerQuestion VirtualMachine.Interac t.DeviceConnection VirtualMachine.Interac t.GuestControl VirtualMachine.Interac t.PowerOff VirtualMachine.Interac t.PowerOn VirtualMachine.Interac t.Reset VirtualMachine.Interac t.SetCDMedia VirtualMachine.Interac t.ToolsInstall	Управление состоянием виртуальных машин, подключением устройств и взаимодействием с гостевой операционной системой.
Virtual Machine > Provisioning	Clone virtual machine, Customize guest, Deploy template, Allow virtual machine download, Allow virtual machine files upload, Read customization specifications	VirtualMachine.Provisi oning.Clone VirtualMachine.Provisi oning.Customize VirtualMachine.Provisi oning.DeployTemplate VirtualMachine.Provisi oning.GetVmFiles VirtualMachine.Provisi oning.PutVmFiles VirtualMachine.Provisi oning.ReadCustSpecs	Клонирование шаблонов виртуальных машин, их настройка и развертывание при создании узлов кластера Deckhouse Kubernetes Platform.
Virtual Machine > Snapshot Management	Create snapshot, Remove Snapshot, Rename Snapshot	VirtualMachine.State.C reateSnapshot VirtualMachine.State.R emoveSnapshot VirtualMachine.State.R enameSnapshot	Управление снимками виртуальных машин и томов в сценариях, где эта функциональность используется компонентами платформы.

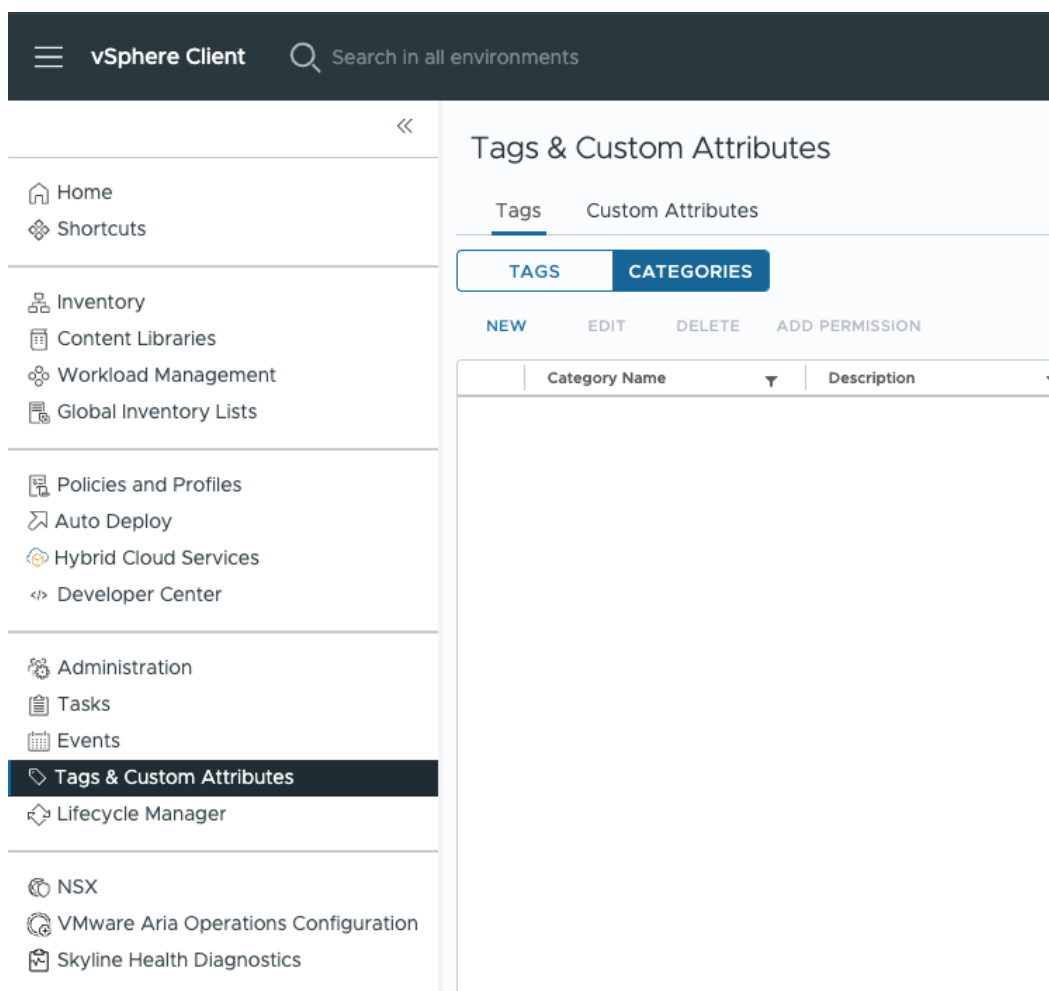
Конфигурация vSphere

Настройка через vSphere Client

Создание тегов и категорий тегов с использованием vSphere Client

В VMware vSphere нет понятий «регион» и «зона». «Регионом» в vSphere является Datacenter, а «зоной» — Cluster. Для создания этой связи используются теги.

1. Откройте vSphere Client и перейдите в «Menu» → «Tags & Custom Attributes» → «Tags».



- Откройте вкладку «Categories» и нажмите «NEW». Создайте категорию для регионов (например `k8s-region`): установите значение «One tag» для параметра «Tags Per Object» и задайте связываемые типы, включая Datacenter.

Create Category ×

Category Name:

Description:

Tags Per Object: ☒ One tag ☐ Many tags

Associable Object Types:

☐ All objects	☐ Cluster
☐ Folder	☐ Datastore
☒ Datacenter	☐ Distributed Port Group
☐ Datastore Cluster	☐ Host
☐ Distributed Switch	☐ Library Item
☐ Content Library	☐ Resource Pool
☐ Network	☐ Virtual Machine
☐ vApp	

- Создайте вторую категорию для зон (например, `k8s-zone`) с типами объектов Host, Cluster и Datastore.

Create Category ×

Category Name:

Description:

Tags Per Object: ☒ One tag ☐ Many tags

Associable Object Types:

☐ All objects
 ☐ Folder
 ☐ Datacenter
 ☐ Datastore Cluster
 ☐ Distributed Switch
 ☐ Content Library
 ☐ Network
 ☐ vApp

☒ Cluster
 ☒ Datastore
 ☐ Distributed Port Group
 ☒ Host
 ☐ Library Item
 ☐ Resource Pool
 ☐ Virtual Machine

4. Перейдите на вкладку «Tags» и создайте минимум по одному тегу в категории региона и в категории зон (например, `test-region`, `test-zone-1`).

☰

vSphere Client

🔍 Search in all environments

«

Home

Shortcuts

Inventory

Content Libraries

Workload Management

Global Inventory Lists

Policies and Profiles

Auto Deploy

Hybrid Cloud Services

Developer Center

Administration

Tasks

Events

Tags & Custom Attributes

Lifecycle Manager

NSX

VMware Aria Operations Configuration

Skyline Health Diagnostics

Tags & Custom Attributes

Tags

Custom Attributes

TAGS

CATEGORIES

NEW

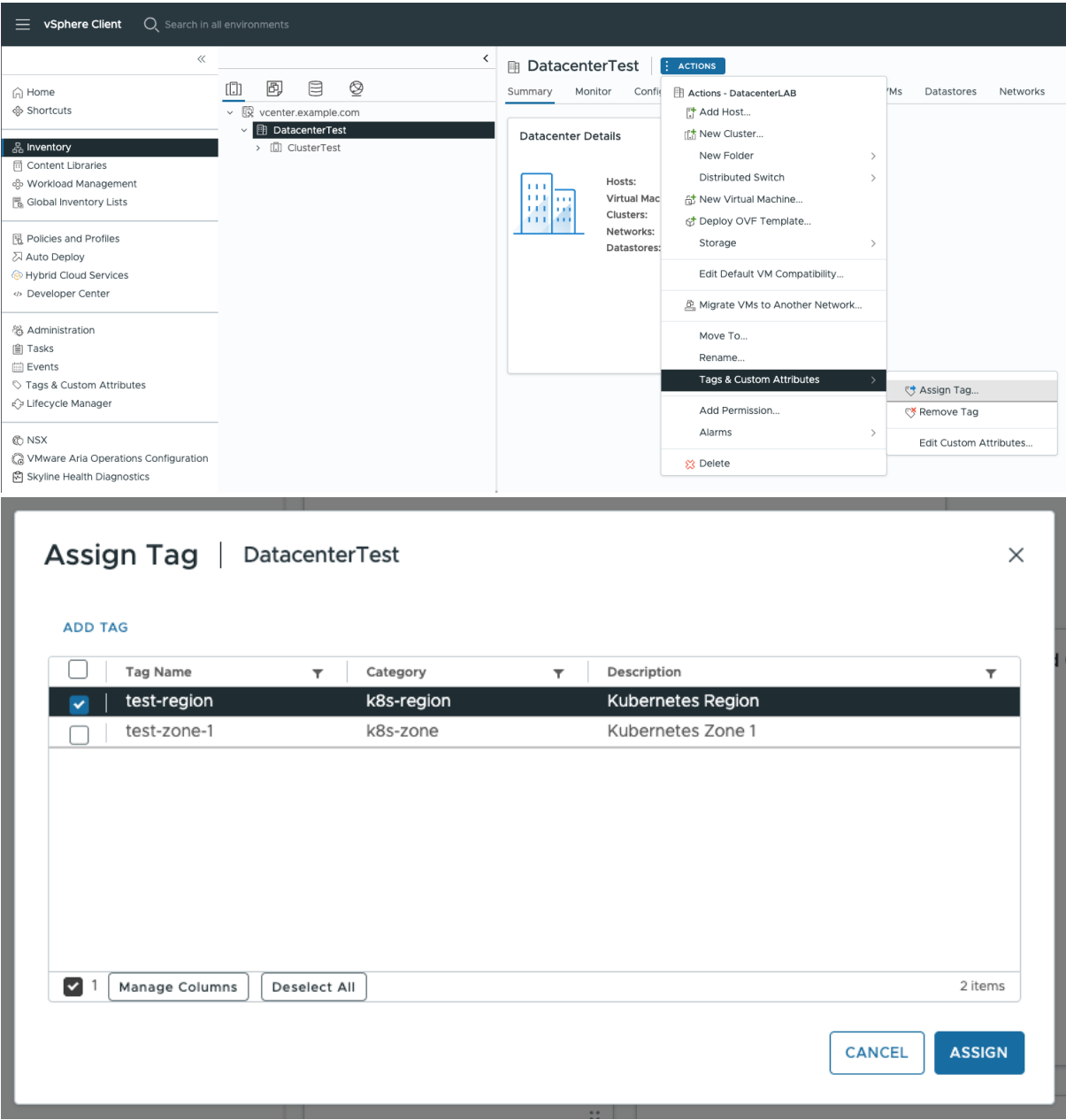
EDIT

DELETE

ADD PERMISSION

	Tag Name	Category
☐	test-region	k8s-region
☐	test-zone-1	k8s-zone

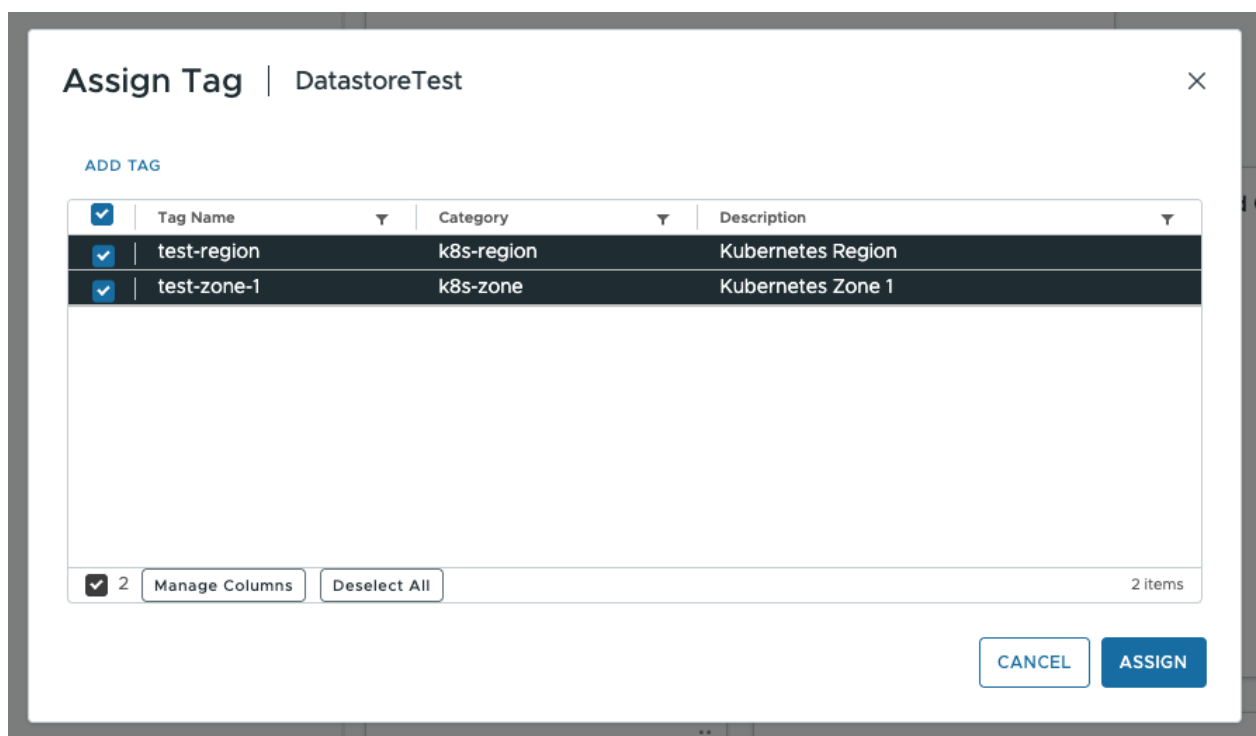
5. Во вкладке «Inventory» выберите целевой Datacenter, перейдите в панель «Summary», откройте «Actions» → «Tags & Custom Attributes» → «Assign Tag» и назначьте тег региона. Повторите для каждого Cluster, на котором будут узлы, назначая соответствующие теги зон.



Настройка Datastore с использованием vSphere Client

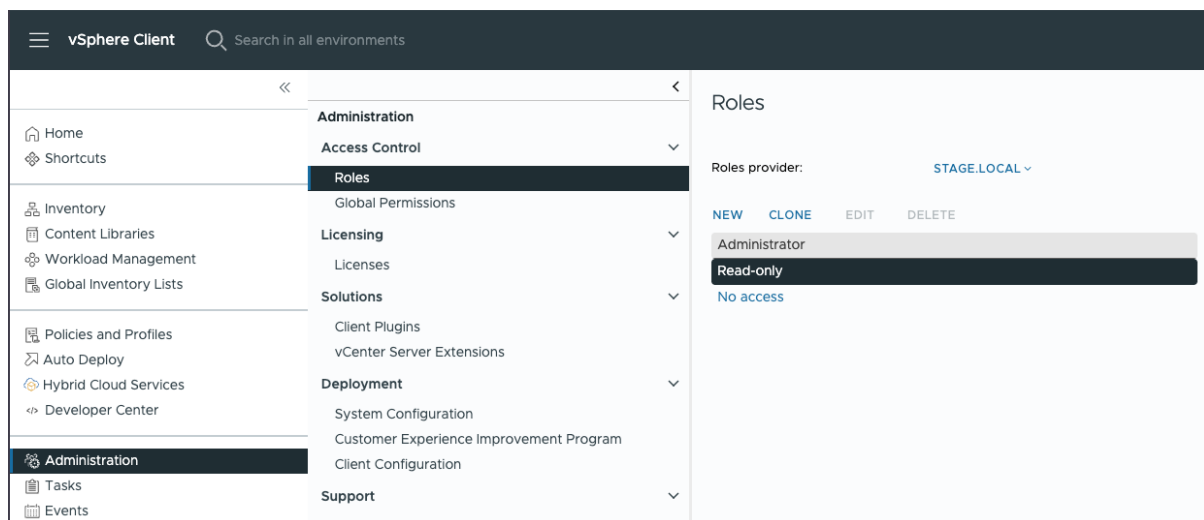
⚠ Для динамического заказа PersistentVolume необходимо, чтобы Datastore был доступен на **каждом** хосте ESXi в зоне (shared datastore).

Во вкладке «Inventory» выберите Datastore, перейдите в панель «Summary», затем откройте меню «Actions» → «Tags & Custom Attributes» → «Assign Tag». Назначьте Datastore тот же тег региона, что и у соответствующего Datacenter, а также тот же тег зоны, что и у соответствующего Cluster.



Создание и назначение роли с использованием vSphere Client

1. Перейдите в «Menu» → «Administration» → «Access Control» → «Roles».



2. Нажмите «NEW», введите имя роли (например, `deckhouse`) и добавьте привилегии из списка.

New Role ✕

Role name
deckhouse

Description

Show All ▾

- Alarms
- AutoDeploy
- Certificate Authority
- Certificate Management
- Certificate Management
- Subscriptions
- Certificates
- ✓ Cns
- Compute Policy
- Content Library

☒ Select all

☒ Searchable

Show All ▾

CANCEL CREATE

3. Назначьте роль для учётной записи Deckhouse, во вкладке «Menu» → «Administration» → «Access Control» → «Global Permissions» нажмите «ADD» и выберите пользователя и роль `deckhouse`.

Add Permission | Global Permission Root ✕

Domain stage.local

User/Group test-user ▾

Role deckhouse

☐ Propagate to children

CANCEL OK

Настройка через govc

Установка govc

Для дальнейшей настройки vSphere потребуется CLI-утилита govc.

После установки задайте переменные окружения для работы с vCenter.



Обязательно указывайте имя пользователя вместе с доменом, например: `username@vsphere.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Создание тегов и категорий тегов с использованием govc

В VMware vSphere нет понятий «регион» и «зона». «Регионом» в vSphere является Datacenter, а «зоной» — Cluster. Для создания этой связи используются теги.

Создайте категории тегов с помощью команд:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Создайте теги в каждой категории. Если вы планируете использовать несколько «зон» (Cluster), создайте тег для каждой из них:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Назначьте тег «региона» на Datacenter:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>
```

Назначьте теги «зон» на объекты Cluster:

```
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
```

Настройка Datastore с использованием govc



Для динамического заказа PersistentVolume необходимо, чтобы Datastore был доступен на **каждом** хосте ESXi (shared datastore).

Для автоматического создания StorageClass в кластере Kubernetes назначьте созданные ранее теги «региона» и «зоны» на объекты Datastore:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Создание и назначение роли с использованием govc



Ввиду разнообразия подключаемых к vSphere SSO-провайдеров шаги по созданию пользователя в данной статье не рассматриваются.

Роль, которую предлагается создать далее, включает в себя привилегии из раздела «Список необходимых привилегий». При необходимости более гранулярных прав обратитесь в техподдержку Deckhouse.

Создайте роль с необходимыми привилегиями:

```
govc role.create deckhouse \
  Cns.Searchable \
  Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
  Folder.Create Folder.Delete Folder.Move Folder.Rename \
  Global.GlobalTag Global.SystemTag \
  InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
  InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
  InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
  InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
  InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
  Network.Assign \
  Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
  StorageProfile.View \
  System.Anonymous System.Read System.View \
  VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
  VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
  VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
  VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
  VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
  VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
  VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
  VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
  VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
  VirtualMachine.Config.UpgradeVirtualHardware \
  VirtualMachine.GuestOperations.Query \
  VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
  VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
  VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
  VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
  VirtualMachine.Inventory.Move \
  VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
  VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles \
  VirtualMachine.Provisioning.ReadCustSpecs \
  VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Назначьте пользователю роль на объекте vCenter.

⚠ Обязательно указывайте имя пользователя вместе с доменом, например: `username@vsphere.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

i Для более детальной настройки прав обратитесь к официальной документации.

Требования к образу виртуальной машины

Для создания шаблона виртуальной машины (Template) рекомендуется использовать готовый cloud-образ/OVA-файл, предоставляемый вендором ОС:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (секция *Generic Cloud / OpenStack*)

i Если вы планируете использовать дистрибутив отечественной ОС, обратитесь к вендору ОС для получения образа/OVA-файла.

⚠ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Подготовка образа виртуальной машины

DKP использует `cloud-init` для настройки виртуальной машины после запуска.



Отключите VMware Guest OS Customization (а также любые механизмы vApp/OS customization, если они применимы в вашей схеме) для шаблона и виртуальных машин кластера. DKP выполняет первичную настройку узлов через `cloud-init` (datasource VMware GuestInfo). Включенная customization может конфликтовать с `cloud-init` и приводить к некорректной инициализации узла.

Чтобы подготовить `cloud-init` и образ VM, выполните следующие действия:

1. Установите необходимые пакеты:

Если используется версия `cloud-init` ниже 21.3 (требуется поддержка VMware GuestInfo):

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
```

Если используется версия `cloud-init` 21.3 и выше:

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init
```

2. Проверьте, что в файле `/etc/cloud/cloud.cfg` установлен параметр `disable_vmware_customization: false`.
3. Убедитесь, что в файле `/etc/cloud/cloud.cfg` указан параметр `default_user`. Он необходим для добавления SSH-ключа при запуске VM.
4. Добавьте datasource VMware GuestInfo — создайте файл `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

```
datasource:
  VMware:
    vmware_cust_file_max_wait: 10
```

5. Перед созданием шаблона VM сбросьте идентификаторы и состояние `cloud-init`, используя следующие команды:

```
truncate -s 0 /etc/machine-id &&
rm /var/lib/dbus/machine-id &&
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

6. Очистите логи событий `cloud-init`:

```
cloud-init clean --logs --seed
```



После запуска виртуальной машины в ней должны быть запущены следующие службы, связанные с пакетами, установленными при подготовке `cloud-init`:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

Чтобы убедиться в том, что службы включены, используйте команду:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Пример ответа для включенных служб:

```
enabled
enabled
enabled
```



DKP создаёт диски виртуальных машин с типом `eagerZeroedThick`, но тип дисков созданных VM будет изменён без уведомления, согласно настроенным в vSphere VM Storage Policy. Подробнее можно прочитать в документации.



DKP использует интерфейс `ens192`, как интерфейс по умолчанию для виртуальных машин в vSphere. Поэтому, при использовании статических IP-адресов в `mainNetwork`, вы должны в образе ОС создать интерфейс с именем `ens192`, как интерфейс по умолчанию.

Инфраструктура

Сети

Для работы кластера необходим VLAN с DHCP и доступом в интернет:

- Если VLAN публичный (публичные адреса), нужна вторая сеть, в которой необходимо развернуть сеть узлов кластера (в этой сети DHCP не нужен).
- Если VLAN внутренний (приватные адреса), эта же сеть будет сетью узлов кластера.

Входящий трафик

- Если у вас имеется внутренний балансировщик запросов, можно обойтись им и направлять трафик напрямую на frontend-узлы кластера.
- Если балансировщика нет, для организации отказоустойчивых LoadBalancer'ов рекомендуется использовать MetalLB в режиме BGP. В кластере будут созданы frontend-узлы с двумя интерфейсами. Для этого дополнительно потребуются:
 - отдельный VLAN для обмена трафиком между BGP-роутерами и MetalLB. В этом VLAN'е должны быть DHCP и доступ в интернет;
 - IP-адреса BGP-роутеров;
 - ASN (номер автономной системы) на BGP-роутере;
 - ASN (номер автономной системы) в кластере;
 - диапазон, из которого анонсировать адреса.

Использование хранилища данных

В кластере может одновременно использоваться различное количество типов хранилищ. В минимальной конфигурации потребуются:

- Datastore, в котором Kubernetes-кластер будет заказывать PersistentVolume;
- Datastore, в котором будут заказываться root-диски для виртуальной машины (это может быть тот же Datastore, что и для PersistentVolume).

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

VsphereInstanceClass

Scope: Cluster

- v1
- v1alpha1

Параметры группы vSphere VirtualMachines, которые будет использовать `machine-controller-manager` (модуль node-manager).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalNetworks**
массив строк

Список путей до сетей, которые будут подключены к виртуальной машине.

Примеры:

```
additionalNetworks: DEVOPS_32
```

```
additionalNetworks: DEVOPS_50
```

- **spec.datastore**
строка

Путь до Datastore (относительно датацентра), на котором будут созданы клонированные виртуальные машины.

По умолчанию используется `datastore` из InstanceClass'a `master`.

Пример:

```
datastore: lun-1201
```

- **spec.disableTimesync**
булевый

Отключение синхронизации времени в гостевой виртуальной машине.



Внимание. Не отключает работу NTP-служб в гостевой ОС, отключает только синхронизацию времени с ESXi.

По умолчанию: `true`

- **spec.mainNetwork**
строка

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Если значение не указано, будет использовано значение указанное в `VsphereClusterConfiguration`.

Пример:

```
mainNetwork: k8s-msk-178
```

- **spec.memory**
целочисленный

Обязательный параметр

Количество памяти (в мегабайтах), выделенной виртуальной машине.

Пример:

```
memory: 8192
```

- **spec.numCPUs**
целочисленный

Обязательный параметр

Количество виртуальных процессорных ядер, выделяемых виртуальной машине.

Пример:

```
numCPUs: 2
```

- **spec.resourcePool**
строка

Путь до Resource Pool, в котором будут созданные клонированные виртуальные машины.

Пример:

```
resourcePool: rp-2012
```

- **spec.rootDiskSize**
целочисленный

Размер (в гигабайтах) корневого диска в виртуальной машине. Если в `template` диск меньше, автоматически произойдет его расширение.

По умолчанию: `20`

Пример:

```
rootDiskSize: 20
```

- **spec.runtimeOptions**
объект

Дополнительные (опциональные) параметры виртуальных машин.

- **spec.runtimeOptions.cpuLimit**
целочисленный

Верхний лимит (в MHz) потребляемой частоты процессоров для создаваемых виртуальных машин.

- **spec.runtimeOptions.cpuReservation**
целочисленный

Величина зарезервированной для виртуальной машины частоты CPU (в MHz).

- **spec.runtimeOptions.cpuShares**
целочисленный

Относительная величина CPU Shares для создаваемых виртуальных машин.

- **spec.runtimeOptions.memoryLimit**

целочисленный

Верхний лимит (в мегабайтах) потребляемой памяти для создаваемых виртуальных машин.

- **spec.runtimeOptions.memoryReservation**

целочисленный

Процент зарезервированной для виртуальной машины памяти в кластере. В процентах относительно `.spec.memory`.

По умолчанию: `80`

Допустимые значения: `0 <= X <= 100`

- **spec.runtimeOptions.memoryShares**

целочисленный

Относительная величина Memory Shares для создаваемых виртуальных машин.

Допустимые значения: `0 <= X <= 100`

- **spec.runtimeOptions.nestedHardwareVirtualization**

булевый

Включение Hardware Assisted Virtualization на созданных виртуальных машинах.

- **spec.template**

строка

Путь до VirtualMachine template, который будет скопирован для создания новой виртуальной машины.

По умолчанию используется `template` из InstanceClass'a `master`.

Пример:

```
template: dev/golden_image
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

Параметры группы vSphere VirtualMachines, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**

объект

Обязательный параметр

- **spec.additionalNetworks**

массив строк

Список путей до сетей, которые будут подключены к виртуальной машине.

Примеры:

```
additionalNetworks: DEVOPS_32
```

```
additionalNetworks: DEVOPS_50
```

- **spec.datastore**

строка

Путь до Datastore (относительно датацентра), на котором будут созданы скопированные виртуальные машины.

По умолчанию используется `datastore` из InstanceClass'a `master`.

Пример:

```
datastore: lun-1201
```

- **spec.disableTimesync**

булевый

Отключение синхронизации времени в гостевой виртуальной машине.



Внимание. Не отключает работу NTP-служб в гостевой ОС, отключает только синхронизацию времени с ESXi.

По умолчанию: `true`

- **spec.mainNetwork**
строка

Путь до сети, которая будет подключена к виртуальной машине как основная (шлюз по умолчанию).

Если значение не указано, будет использовано значение указанное в `VsphereClusterConfiguration`.

Пример:

```
mainNetwork: k8s-msk-178
```

- **spec.memory**
целочисленный

Обязательный параметр

Количество памяти (в мегабайтах), выделенной виртуальной машине.

Пример:

```
memory: 8192
```

- **spec.numCPUs**
целочисленный

Обязательный параметр

Количество виртуальных процессорных ядер, выделяемых виртуальной машине.

Пример:

```
numCPUs: 2
```

- **spec.resourcePool**
строка

Путь до Resource Pool, в котором будут созданные клонированные виртуальные машины.

Пример:

```
resourcePool: rp-2012
```

- **spec.rootDiskSize**
целочисленный

Размер (в гигабайтах) корневого диска в виртуальной машине. Если в template диск меньше, автоматически произойдет его расширение.

По умолчанию: `20`

Пример:

```
rootDiskSize: 20
```

- **spec.runtimeOptions**
объект

Дополнительные (опциональные) параметры виртуальных машин.

- **spec.runtimeOptions.cpuLimit**
целочисленный

Верхний лимит (в MHz) потребляемой частоты процессоров для создаваемых виртуальных машин.

- **spec.runtimeOptions.cpuReservation**
целочисленный

Величина зарезервированной для виртуальной машины частоты CPU (в MHz).

- **spec.runtimeOptions.cpuShares**
целочисленный

Относительная величина CPU Shares для создаваемых виртуальных машин.

- **spec.runtimeOptions.memoryLimit**
целочисленный

Верхний лимит (в мегабайтах) потребляемой памяти для создаваемых виртуальных машин.

- **spec.runtimeOptions.memoryReservation**

целочисленный

Процент зарезервированной для виртуальной машины памяти в кластере. В процентах относительно `spec.memory`.

По умолчанию: 80

Допустимые значения: `0 <= X <= 100`

- **spec.runtimeOptions.memoryShares**

целочисленный

Относительная величина Memory Shares для создаваемых виртуальных машин.

Допустимые значения: `0 <= X <= 100`

- **spec.runtimeOptions.nestedHardwareVirtualization**

булевый

Включение Hardware Assisted Virtualization на созданных виртуальных машинах.

- **spec.template**

строка

Путь до VirtualMachine template, который будет скопирован для создания новой виртуальной машины.

По умолчанию используется `template` из InstanceClass'a `master`.

Пример:

```
template: dev/golden_image
```

- **status**

объект

- **status.nodeGroupConsumers**

массив строк

Примеры

Ниже представлен пример конфигурации облачного провайдера VMware vSphere.

Пример конфигурации

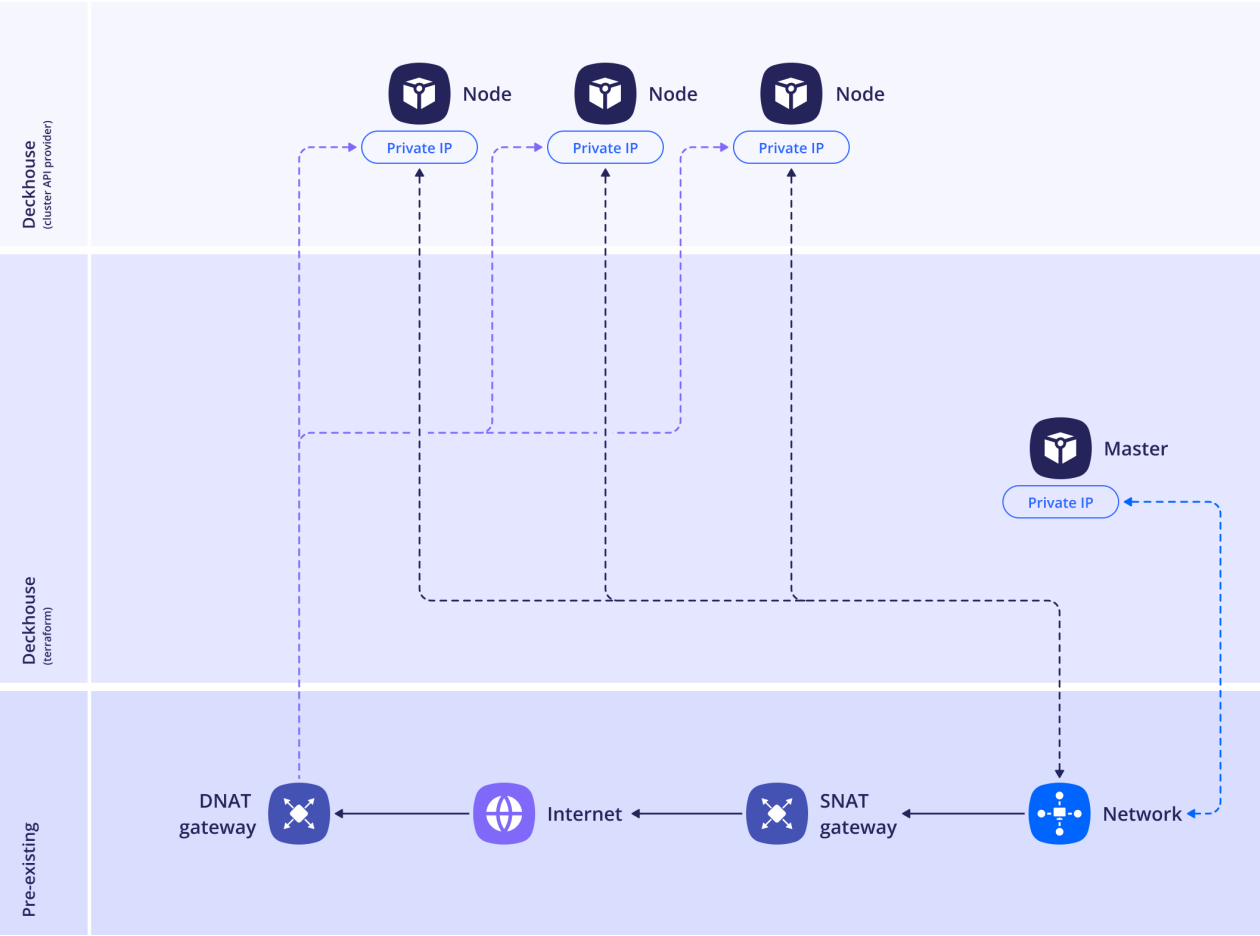
```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: vc-3.internal
    username: user
    password: password
    vmFolderPath: dev/test
    insecure: true
    region: moscow-x001
    sshKeys:
      - "ssh-rsa AAAAB3N...6xHJwwj"
    externalNetworkNames:
      - KUBE-3
      - devops-internal
    internalNetworkNames:
      - KUBE-3
      - devops-internal
  nsxt:
    defaultIpPoolName: "External IP Pool"
    tier1GatewayPath: flant_tier1
    user: guestuser1
    password: pass
    host: 1.2.3.4
    insecureFlag: true
    size: SMALL
```

Пример custom resource VsphereInstanceClass

```
apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: test
spec:
  numCPUs: 2
  memory: 2048
  rootDiskSize: 20
  template: dev/golden_image
  mainNetwork: k8s-msk-178
  datastore: <DS_PARENT_FOLDER>/lun-1201 # <DS_PARENT_FOLDER> – родительская папка Datastore.
```

Cloud provider - VMware vSphere: схемы размещения

Standard



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
    - ru-central1-a
    - ru-central1-b
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
nodeGroups:
- name: khm
  replicas: 1
  zones:
    - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- ru-central1-a
- ru-central1-b

```

Cloud provider — Yandex Cloud

Модуль `cloud-provider-yandex` обеспечивает интеграцию Deckhouse Kubernetes Platform с Yandex Cloud. Он предоставляет возможность модулю `node-manager` использовать ресурсы Yandex Cloud при заказе узлов для `NodeGroup`.

Функции модуля `cloud-provider-yandex`:

- Управление ресурсами Yandex Cloud через `cloud-controller-manager`:
 - создаёт сетевые маршруты для сети `PodNetwork` на стороне Yandex Cloud;
 - создаёт Network Load Balancer и целевые группы для Service с типом LoadBalancer;
 - актуализирует метаданные инстансов и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в Yandex Cloud.
- Заказ дисков через CSI-драйвер Yandex (`yandex.csi.flant.com`) и создание StorageClass для типов дисков Yandex Cloud, чтобы из кластера можно было заказывать PersistentVolume.
- Заказ базовой инфраструктуры и CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-yandex`.
- Заказ CloudEphemeral-узлов через Machine Controller Manager (MCM) или Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе `YandexInstanceClass`.
- Регистрация в модуле `node-manager`, чтобы `YandexInstanceClass` можно было указывать при описании `NodeGroup`.
- Автоматическое включение CNI для новых кластеров. Начиная с DKP 1.76 по умолчанию используется `cni-cilium`.



Модуль находится в процессе миграции управления CloudEphemeral-узлами с Machine Controller Manager (MCM) на Cluster API (CAPI). Существующие `NodeGroup` продолжают использовать MCM, а новые по умолчанию создаются с использованием CAPI. Порядок миграции существующих групп — в разделе «Как мигрировать группы узлов на Cluster API (CAPI)».

Настройки

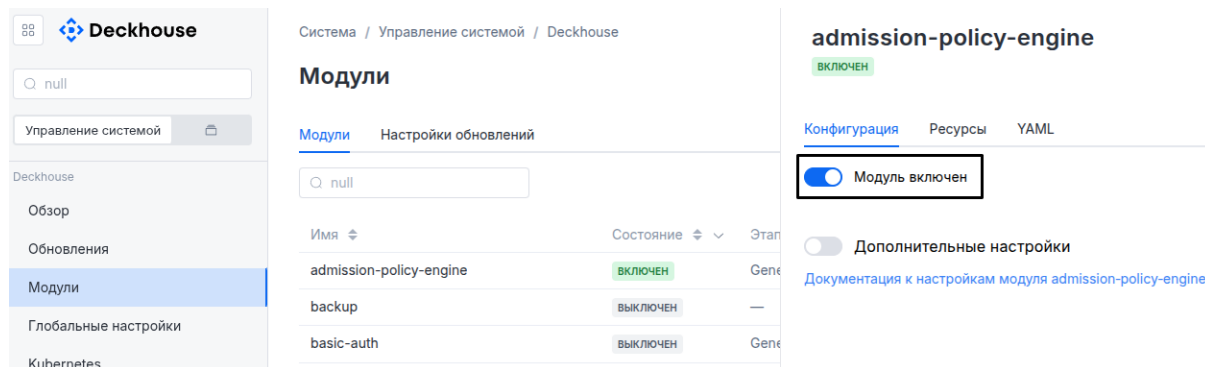
Внимание. При изменении настроек модуля **пересоздания существующих объектов Machines в кластере НЕ происходит** (новые объекты Machine будут создаваться с новыми параметрами). Пересоздание происходит только при изменении параметров NodeGroup и YandexInstanceClass. Подробнее в документации модуля node-manager.

В модуле реализовано 3 алерта.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-yandex`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-yandex`:

```
d8 system module enable cloud-provider-yandex
```

- **С помощью ModuleConfig `cloud-provider-yandex`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-yandex` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-yandex`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-yandex`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cloud-provider-yandex` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-yandex`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-yandex` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-yandex
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-yandex`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-yandex`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Storage

Модуль автоматически создает StorageClass'ы, покрывающие все варианты дисков в Yandex Cloud:

Тип	Имя StorageClass	Комментарии
network-hdd	network-hdd	
network-ssd	network-ssd	
network-ssd-nonreplicated	network-ssd-nonreplicated	
network-ssd-io-m3	network-ssd-io-m3	Размер дисков должен быть кратен 93 ГБ

Вы можете отфильтровать ненужные StorageClass'ы с помощью параметра `exclude`.

LoadBalancer

Модуль подписывается на объекты Service с типом `LoadBalancer` и создает соответствующие `NetworkLoadBalancer` и `TargetGroup` в Yandex Cloud.

Больше информации в документации Kubernetes Cloud Controller Manager for Yandex Cloud.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.additionalExternalNetworkIDs**

массив строк

Список Network ID, которые будут считаться `ExternalIP` при перечислении адресов у Node.

По умолчанию: `[]`

- **settings.storageClass**

объект

- **settings.storageClass.default**

Параметр устарел

строка

Имя StorageClass'а, который будет использоваться по умолчанию в кластере.

Если значение не указано, фактический StorageClass по умолчанию будет определен следующим образом:

- присутствующий StorageClass в кластере, имеющий default-аннотацию;
- первый (по алфавиту) StorageClass из тех, что создаются модулем.

i

Параметр устарел. Вместо этого параметра используйте глобальный параметр `global.defaultClusterStorageClass`.

Пример:

```
default: network-hdd
```

- **settings.storageClass.exclude**

массив строк

Полные имена (или regex-выражения имен) StorageClass, которые не будут созданы в кластере.

Пример:

```
exclude:
- network-ssd-.*
- network-hdd
```

Настройки провайдера

YandexClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Yandex Cloud.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```

i

После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
nodeNetworkCIDR: 127.0.0.1/8
labels:
  label-2: b
sshPublicKey: "<SSH_PUBLIC_KEY>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
    coreFraction: 50
    externalIPAddresses:
    - 198.51.100.5
    - Auto
provider:
  cloudID: "<CLOUD_ID>"
  folderID: "<FOLDER_ID>"
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJtBLt+\n-----END PRIVATE KEY-----\n"
    }

```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **dhcpOptions**

объект

Список DHCP-опций, которые будут установлены на все подсети.

Возможные проблемы при использовании.

- **dhcpOptions.domainName**

строка

Search-домен.

- **dhcpOptions.domainNameServers**

массив строк

Список адресов рекурсивных DNS.

- **Элемент массива**

строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **existingNetworkID**

строка

ID существующей VPC Network.

- **existingZoneToSubnetIDMap**

объект

Одна или несколько ранее существовавших подсетей, сопоставленных с соответствующей зоной.

Внимание. При использовании `cni-simple-bridge` DKP создаст таблицу маршрутизации, которую необходимо вручную привязать к указанным подсетям. К одной подсети можно привязать только одну

таблицу маршрутизации, поэтому невозможно развернуть несколько кластеров с `cni-simple-bridge` в одних и тех же подсетях. Начиная с DKP 1.76 для новых кластеров в Yandex Cloud по умолчанию используется `cni-cilium` в режиме `VXLAN`. В этом режиме маршрутизация трафика подов не зависит от таблиц маршрутизации Yandex Cloud.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Пример:

```
existingZoneToSubnetIDMap:
  ru-central1-a: e2lu8r1tbbtryhdpa9ro
  ru-central1-b: e2lu8r1tbbtryhdpa9ro
  ru-central1-d: e2lu8r1tbbtryhdpa9ro
```

- **kind**

строка

Обязательный параметр

Допустимые значения: `YandexClusterConfiguration`

- **labels**

объект

Лейблы, проставляемые на ресурсы, создаваемые в Yandex Cloud.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

Допустимые значения: `Standard`, `WithoutNAT`, `WithNATInstance`

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.



Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

- **masterNodeGroup.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей `YandexInstanceClass`.

- **masterNodeGroup.instanceClass.additionalLabels**

объект

Дополнительные лейблы, которые будут присвоены созданным экземплярам.

Пример:

```
project: cms-production
severity: critical
```

- **masterNodeGroup.instanceClass.cores**

целочисленный

Обязательный параметр

Количество ядер у создаваемых экземпляров.

- **masterNodeGroup.instanceClass.diskSizeGB**

целочисленный

Размер диска у экземпляров. Значение указывается в `Гиб`.

По умолчанию: `50`

Пример:

```
diskSizeGB: 0
```

■ **masterNodeGroup.instanceClass.diskType**

строка

Тип диска у создаваемых инстансов. Размер дисков `network-ssd-nonreplicated` и `network-ssd-io-m3` должен быть кратен 93 GB.

По умолчанию: `network-ssd`

Допустимые значения: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

Пример:

```
diskType: network-ssd-io-m3
```

■ **masterNodeGroup.instanceClass.etcdDiskSizeGb**

целочисленный

Размер диска для etcd. Значение указывается в ГиБ.

При изменении этого параметра необходимо вручную расширить каждый диск для etcd в интерфейсе Yandex Cloud (причина такого поведения — в [issue](#)).

По умолчанию: `10`

Пример:

```
etcdDiskSizeGb: 10
```

■ **masterNodeGroup.instanceClass.externalIPAddresses**

массив строк

Список внешних IP-адресов, перечисленных в порядке зон, в которых будут создаваться узлы.

В списке можно указывать следующие значения:

- IP-адрес из дополнительной внешней сети для соответствующей зоны (параметр `externalSubnetIDs`);
- зарезервированный публичный IP-адрес, если список дополнительных внешних сетей не определен (параметр `externalSubnetIDs`);
- `Auto`, для заказа публичного IP-адреса в соответствующей зоне.

Обратите внимание:

- Элементы массива должны быть перечислены согласно порядку зон (параметр `zones`), в которых будут создаваться узлы.

Например:

```
- <RU-CENTRAL1-A-IP-ADDRESS>
- <RU-CENTRAL1-B-IP-ADDRESS>
- <RU-CENTRAL1-D-IP-ADDRESS>
```

- Количество элементов массива в параметре `externalIPAddresses` должно быть равно количеству узлов в группе (параметра `replicas`). Для автоматического заказа публичных IP-адресов в зоне, используйте значение `Auto` в списке в параметре `externalIPAddresses` на соответствующем месте (соответствующему порядку зон).
- Если не указать `externalIPAddresses`, то узлы в группе будут заказаны без внешних IP-адресов.

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}|(Auto)$`

■ **masterNodeGroup.instanceClass.externalSubnetID**

Параметр устарел
строка

Подключаемый к узлу дополнительный сетевой интерфейс, в который будет идти маршрут по умолчанию.

■ **masterNodeGroup.instanceClass.externalSubnetIDs**

массив строк

Список дополнительных внешних сетей, подключаемых к узлам.

Интерфейс узла, находящийся в сети из списка, будет назначен шлюзом по умолчанию.

Также будет добавлен маршрут для интерфейса внутренней сети узла на всю подсеть, указанную в `nodeNetworkCIDR`.

Элементы массива **должны** быть перечислены согласно порядку зон (параметр `zones`), в которых будут создаваться узлы. Например:

```
- <RU-CENTRAL1-A-SUBNET-ID>
- <RU-CENTRAL1-B-SUBNET-ID>
- <RU-CENTRAL1-D-SUBNET-ID>
```

- **masterNodeGroup.instanceClass.imageID**

строка

Обязательный параметр

Идентификатор образа, который будет установлен в заказанные инстансы.

Пример:

```
imageID: fd83ica41cade1mj35sr
```

- **masterNodeGroup.instanceClass.memory**

целочисленный

Обязательный параметр

Количество оперативной памяти (в мегабайтах) у создаваемых инстансов.

Пример:

```
memory: 8192
```

- **masterNodeGroup.instanceClass.networkType**

строка

Тип сети.

По умолчанию: `Standard`

Допустимые значения: `Standard`, `SoftwareAccelerated`

- **masterNodeGroup.instanceClass.platform**

строка

Платформа виртуальной машины.

По умолчанию: `standard-v2`

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **masterNodeGroup.zones**

массив

Необязательный параметр.

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

- **Элемент массива**

Допустимые значения: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

- **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей YandexInstanceClass.

- **nodeGroups.instanceClass.additionalLabels**

объект

Дополнительные лейблы, которые будут присвоены созданным инстансам.

Пример:

```
project: cms-production
severity: critical
```

■ **nodeGroups.instanceClass.coreFraction**
целочисленный

Базовый уровень производительности каждого ядра CPU у создаваемых инстансов.
Подробнее...

По умолчанию: 100

Допустимые значения: 5, 20, 50, 100

Пример:

```
coreFraction: 20
```

■ **nodeGroups.instanceClass.cores**
целочисленный

Обязательный параметр

Количество ядер у создаваемых инстансов.

■ **nodeGroups.instanceClass.diskSizeGB**
целочисленный

Размер диска у инстансов. Значение указывается в Гиб.

По умолчанию: 50

Пример:

```
diskSizeGB: 0
```

■ **nodeGroups.instanceClass.diskType**
строка

Тип диска у создаваемых инстансов. Размер дисков `network-ssd-nonreplicated` и `network-ssd-io-m3` должен быть кратен 93 GB.

По умолчанию: `network-ssd`

Допустимые значения: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

Пример:

```
diskType: network-ssd-io-m3
```

■ **nodeGroups.instanceClass.externalIPAddresses**
массив строк

Список внешних IP-адресов, перечисленных в порядке зон, в которых будут создаваться узлы.

В списке можно указывать следующие значения:

- IP-адрес из дополнительной внешней сети для соответствующей зоны (параметр `externalSubnetIDs`);
- зарезервированный публичный IP-адрес, если список дополнительных внешних сетей не определен (параметр `externalSubnetIDs`);
- `Auto`, для заказа публичного IP-адреса в соответствующей зоне.

Обратите внимание:

- Элементы массива должны быть перечислены согласно порядку зон (параметр `zones`), в которых будут создаваться узлы.

Например:

```
- <RU-CENTRAL1-A-IP-ADDRESS>
- <RU-CENTRAL1-B-IP-ADDRESS>
- <RU-CENTRAL1-D-IP-ADDRESS>
```

- Количество элементов массива в параметре `externalIPAddresses` должно быть равно количеству узлов в группе (параметра `replicas`). Для автоматического заказа публичных IP-адресов в зоне, используйте значение `Auto` в списке в параметре `externalIPAddresses` на соответствующем месте (соответствующему порядку зон).
- Если не указать `externalIPAddresses`, то узлы в группе будут заказаны без внешних IP-адресов.

■ **Элемент массива**
строка

Шаблон: `^([0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}) | (Auto)$`

- **nodeGroups.instanceClass.externalSubnetID**

Параметр устарел
строка

Подключаемый к узлу дополнительный сетевой интерфейс, в который будет идти маршрут по умолчанию.

- **nodeGroups.instanceClass.externalSubnetIDs**

массив строк

Список дополнительных внешних сетей, подключаемых к узлам.

Интерфейс узла, находящийся в сети из списка, будет назначен шлюзом по умолчанию.

Также будет добавлен маршрут для интерфейса внутренней сети узла на всю подсеть, указанную в nodeNetworkCIDR.

Элементы массива **должны** быть перечислены согласно порядку зон (параметр `zones`), в которых будут создаваться узлы. Например:

```
- <RU-CENTRAL1-A-SUBNET-ID>
- <RU-CENTRAL1-B-SUBNET-ID>
- <RU-CENTRAL1-D-SUBNET-ID>
```

- **nodeGroups.instanceClass.imageID**

строка

Обязательный параметр

Идентификатор образа, который будет установлен в заказанные инстансы.

Пример:

```
imageID: fd83ica4lcade1mj35sr
```

- **nodeGroups.instanceClass.memory**

целочисленный

Обязательный параметр

Количество оперативной памяти (в мегабайтах) у создаваемых инстансов.

Пример:

```
memory: 8192
```

- **nodeGroups.instanceClass.networkType**

строка

Тип сети.

По умолчанию: `Standard`

Допустимые значения: `Standard`, `SoftwareAccelerated`

- **nodeGroups.instanceClass.platform**

строка

Платформа виртуальной машины.

По умолчанию: `standard-v2`

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узла.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: "true"
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять теги в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

■ **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

■ **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

■ **nodeGroups.nodeTemplate.taints.key**

строка

■ **nodeGroups.nodeTemplate.taints.value**

строка

○ **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

○ **nodeGroups.zones**

массив

Список зон, в которых допустимо создавать узлы.

Порядок указания зон не влияет на порядок создания узлов. Узлы будут размещаться в зонах в алфавитном порядке: первый узел — в зоне, стоящей первой по алфавиту, второй — во второй зоне и так далее. Если количество зон меньше числа узлов, то после размещения узла в последней зоне по алфавиту распределение продолжится сначала списка.

■ **Элемент массива**

Допустимые значения: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

● **nodeNetworkCIDR**

строка

Обязательный параметр

Данная подсеть будет разделена на **три** равные части и использована для создания подсетей в трех зонах Yandex Cloud.

● **provider**

объект

Обязательный параметр

Параметры подключения к API Yandex Cloud.

○ **provider.cloudID**

строка

Обязательный параметр

Идентификатор облака.

○ **provider.folderID**

строка

Обязательный параметр

Идентификатор директории.

- **provider.serviceAccountJSON**
строка

Обязательный параметр

Ключ к service account'у в JSON-формате.

Получить его можно с помощью команды `[yc iam key create](environment.html)`.

Шаблон: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

Пример:

```
serviceAccountJSON: |
{
  "id": "...",
  "service_account_id": "...",
  "created_at": "2022-08-04T05:38:34.756137618Z",
  "key_algorithm": "RSA_2048",
  "public_key": "-----BEGIN PUBLIC KEY-----.....END PUBLIC KEY-----\n",
  "private_key": "-----BEGIN PRIVATE KEY-----.....END PRIVATE KEY-----\n"
}
```

- **sshPublicKey**
строка

Обязательный параметр

Публичный ключ для доступа на узлы.

- **withNATInstance**
объект

Настройки для схемы размещения `WithNATInstance`.

- **withNATInstance.exporterAPIKey**
строка

API-ключ для экспортера метрик Yandex Cloud.

- Если значение ключа — пустая строка, экспортер не будет развернут в кластере.
- Если значение ключа — `Auto`, Deckhouse создаст service account с ролью `monitoring.viewer` и API-ключ для него. Для основного service account'a требуется роль `admin`.
- Любое другое значение ключа считается допустимым API-ключем. Используйте инструкцию для создания API-ключа. Service account'у, для которого будет создан ключ, необходимо назначить роль `monitoring.viewer`.

По умолчанию:

- **withNATInstance.externalSubnetID**
строка

Подключаемый к узлу дополнительный сетевой интерфейс, в который будет идти маршрут по умолчанию.

- **withNATInstance.internalSubnetCIDR**

CIDR автоматически создаваемой подсети для внутреннего интерфейса. Если указан вместе с `internalSubnetID`, `internalSubnetCIDR` имеет приоритет.

- **withNATInstance.internalSubnetID**
строка

ID подсети для внутреннего интерфейса.

- **withNATInstance.natInstanceExternalAddress**
строка

Внешний зарезервированный IP-адрес или адрес из `externalSubnetID` при указании опции.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **withNATInstance.natInstanceInternalAddress**
Параметр устарел
строка

Лучше не использовать эту опцию, а использовать автоматически назначаемые адреса.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **withNATInstance.natInstanceResources**
объект

Вычислительные ресурсы, выделяемые для NAT-инстанса. Если параметр не указан, будут использоваться значения по умолчанию.

Внимание. При изменении этих параметров, `terraform-auto-converger` перезапустит машину NAT-инстанса автоматически, если включена настройка `autoConvergerEnabled`. Это может привести к временному прерыванию трафика в кластере.

■ **withNATInstance.natInstanceResources.cores**

целочисленный

Количество ядер у создаваемого NAT-инстанса.

По умолчанию: 2

■ **withNATInstance.natInstanceResources.memory**

целочисленный

Количество оперативной памяти (в мегабайтах) у создаваемого NAT-инстанса.

По умолчанию: 2048

■ **withNATInstance.natInstanceResources.platform**

строка

Тип платформы процессора у создаваемого NAT-инстанса.

По умолчанию: `standard-v2`

• **zones**

массив

Глобальное ограничение набора зон, с которыми работает данный cloud-провайдер.

◦ **Элемент массива**

Допустимые значения: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

Подготовка окружения



Чтобы использовать `ContainerdV2` в качестве container runtime на узлах кластера, ОС на виртуальных машинах должны соответствовать следующим требованиям:

- ядро Linux версии 5.8 и новее, за исключением диапазонов 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- поддержка `CgroupsV2`;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.

Подробнее — в документации.



На виртуальных машинах должен быть установлен пакет `cloud-init`. После запуска виртуальной машины должны быть запущены следующие службы, связанные с этим пакетом:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.



Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Создание сервисного аккаунта

Чтобы Deckhouse Kubernetes Platform (DKP) могла управлять ресурсами в облаке Yandex Cloud, необходимо создать сервисный аккаунт и выдать ему права на редактирование. Подробная инструкция по созданию сервисного аккаунта в Yandex Cloud доступна в документации провайдера. Далее представлена краткая последовательность необходимых действий:

1. Создайте пользователя с именем `deckhouse`:

```
yc iam service-account create --name deckhouse
```

В ответ вернутся параметры пользователя:

```
id: <userID>
folder_id: <folderID>
created_at: "YYYY-MM-DDTHH:MM:SSZ"
name: deckhouse
```


2. Назначьте необходимые роли вновь созданному пользователю для своего облака:

```
yc resource-manager folder add-access-binding --id <folderID> --role compute.editor --subject serviceAccount:
<userID>
yc resource-manager folder add-access-binding --id <folderID> --role vpc.admin --subject serviceAccount:
<userID>
yc resource-manager folder add-access-binding --id <folderID> --role load-balancer.editor --subject
serviceAccount:<userID>
```

3. Создайте JSON-файл с параметрами авторизации пользователя в облаке. В дальнейшем с помощью этих данных будет происходить авторизация в облаке:

```
yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
```



Может потребоваться увеличение квот.

При необходимости зарезервируйте публичный IP-адрес.

Квоты

При заказе нового кластера необходимо увеличить квоты в консоли Yandex Cloud.

Рекомендованные значения квот при создании нового кластера:

- Количество виртуальных процессоров: 64.
- Общий объем SSD-дисков: 2000 ГБ.
- Количество виртуальных машин: 25.
- Общий объем RAM виртуальных машин: 256 ГБ.

Интеграция с Yandex Cloud

Начиная с версии DKP 1.76, в Yandex Cloud CNI `cilium` используется по умолчанию для новых кластеров.

При использовании CentOS 8 необходимо явно включить модуль `cni-simple-bridge` в файле `config.yml`.

Пример конфигурации CNI для CentOS 8 в Yandex Cloud...

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-simple-bridge
spec:
  enabled: true
```

Настройка групп безопасности

При создании облачной сети, Yandex Cloud создаёт группу безопасности по умолчанию для всех подключенных сетей, включая сеть кластера Deckhouse Kubernetes Platform. Эта группа безопасности по умолчанию содержит правила разрешающие любой входящий и исходящий трафик и применяется для всех подсетей облачной сети, если на объект (интерфейс ВМ) явно не назначена другая группа безопасности.



Не удаляйте правила по умолчанию, разрешающие любой трафик, до того как закончите настройку правил группы безопасности. Это может нарушить работоспособность кластера.

Ниже приведены общие рекомендации по настройке группы безопасности. Некорректная настройка групп безопасности может сказаться на работоспособности кластера. Пожалуйста, ознакомьтесь с особенностями работы групп безопасности в Yandex Cloud перед использованием в продуктивных средах.

1. Определите облачную сеть, в которой работает кластер Deckhouse Kubernetes Platform.

Название сети совпадает с полем `prefix` ресурса `ClusterConfiguration`. Его можно узнать с помощью команды:

```
d8 k get secrets -n kube-system d8-cluster-configuration -ojson | \
jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
```

2. В консоли Yandex Cloud выберите сервис Virtual Private Cloud и перейдите в раздел *Группы безопасности*. У вас должна отображаться одна группа безопасности с пометкой `Default`.

Группы безопасности

Фильтровать по имени или идентификатору

Имя	Идентификатор	Описание	Статус
default-sg-enpcnh1tm19i7722oap3	Default enprl7sg8f7u0eglcmls	Default security group for network	Active

3. Создайте правила согласно инструкции Yandex Cloud.

Правила

Входящий трафик Исходящий трафик

Протокол	Диапазон портов	Тип источника	Источник	Описание	
TCP	0-65535	Группа безопасности	Balancer	—	...
Any	0-65535	Группа безопасности	Self	—	...
Any	—	CIDR	10.111.0.0/16	podSubnetCIDR	...
Any	—	CIDR	10.222.0.0/16	serviceSubnetCIDR	...
ICMP	—	CIDR	10.0.0.0/8	+2	...
TCP	30000-32767	CIDR	0.0.0.0/0	—	...
TCP	22	CIDR	0.0.0.0/0	—	...

Добавить правило

Сохранить

4. Удалите правило, разрешающее любой **входящий** трафик (на скриншоте выше оно уже удалено), и сохраните изменения.

Интеграция с Yandex Lockbox

С помощью инструмента External Secrets Operator вы можете настроить синхронизацию секретов Yandex Lockbox с секретами кластера Deckhouse Kubernetes Platform.

Приведенную инструкцию следует рассматривать как *Быстрый старт*. Для использования интеграции в продуктивных средах ознакомьтесь со следующими ресурсами:

- Yandex Lockbox
- Синхронизация с секретами Yandex Lockbox
- External Secrets Operator

Инструкция по развертыванию

1. Создайте сервисный аккаунт, необходимый для работы External Secrets Operator:

```
yc iam service-account create --name eso-service-account
```

2. Создайте авторизованный ключ для сервисного аккаунта и сохраните его в файл:

```
yc iam key create --service-account-name eso-service-account --output authorized-key.json
```

3. Назначьте сервисному аккаунту роли `lockbox.editor`, `lockbox.payloadViewer` и `kms.keys.encrypterDecrypter` для доступа ко всем секретам каталога:

```
folder_id=<идентификатор каталога>
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --role lockbox.editor
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --role lockbox.payloadViewer
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --role kms.keys.encrypterDecrypter
```

Для более тонкой настройки ознакомьтесь с управлением доступом в Yandex Lockbox.

4. Установите External Secrets Operator с помощью Helm-чарта согласно инструкции.

Обратите внимание, что вам может понадобиться задать `nodeSelector`, `tolerations` и другие параметры. Для этого используйте файл `./external-secrets/values.yaml` после распаковки Helm-чарта.

Скачайте и распакуйте чарт:

```
helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
--version 0.5.5 \
--untar
```

Установите Helm-чарт:

```
helm install -n external-secrets --create-namespace \
--set-file auth.json=authorized-key.json \
external-secrets ./external-secrets/
```

Где:

- `authorized-key.json` — название файла с авторизованным ключом из шага 2.

5. Создайте хранилище секретов SecretStore, содержащее секрет `sa-creds`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: SecretStore
metadata:
  name: secret-store
spec:
  provider:
    yandexlockbox:
      auth:
        authorizedKeySecretRef:
          name: sa-creds
          key: key
EOF
```

Где:

- `sa-creds` — название `Secret`, содержащий авторизованный ключ. Этот секрет должен появиться после установки Helm-чарта.
- `key` — название ключа в поле `.data` секрета выше.

Проверка работоспособности

1. Проверьте статус External Secrets Operator и созданного хранилища секретов:

```
$ d8 k -n external-secrets get po
NAME                                     READY   STATUS    RESTARTS   AGE
external-secrets-55f78c44cf-dbf6q      1/1     Running   0           77m
external-secrets-cert-controller-78cbc7d9c8-rszhx  1/1     Running   0           77m
external-secrets-webhook-6d7b66758-s7v9c  1/1     Running   0           77m

$ d8 k -n external-secrets get secretstores.external-secrets.io
NAME          AGE   STATUS
secret-store  69m   Valid
```

2. Создайте секрет Yandex Lockbox со следующими параметрами:

- **Имя** — `lockbox-secret`.
- **Ключ** — введите неконфиденциальный идентификатор `password`.
- **Значение** — введите конфиденциальные данные для хранения `p@$w0rd`.

3. Создайте объект ExternalSecret, указывающий на секрет `lockbox-secret` в хранилище `secret-store`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
    - secretKey: password
      remoteRef:
        key: <ИДЕНТИФИКАТОР_СЕКРЕТА>
        property: password
EOF
```

Где:

- `spec.target.name` — имя нового секрета. External Secrets Operator создаст этот секрет в кластере Deckhouse Kubernetes Platform и поместит в него параметры секрета Yandex Lockbox `lockbox-secret`.
- `spec.data[].secretKey` — название ключа в поле `.data` секрета, который создаст External Secrets Operator.
- `spec.data[].remoteRef.key` — идентификатор созданного ранее секрета Yandex Lockbox `lockbox-secret`. Например, `e6q28nvfmhu539*****`.
- `spec.data[].remoteRef.property` — **ключ**, указанный ранее, для секрета Yandex Lockbox `lockbox-secret`.

4. Убедитесь, что новый ключ `k8s-secret` содержит значение секрета `lockbox-secret`:

```
d8 k -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

В выводе команды будет содержаться **значение** ключа `password` секрета `lockbox-secret`, созданного ранее:

```
p@$w0rd
```

Интеграция с Yandex Managed Service for Prometheus

С помощью данной интеграции вы можете использовать Yandex Managed Service for Prometheus в качестве внешнего хранилища метрик, например, для долгосрочного хранения.

Запись метрик

1. Создайте сервисный аккаунт с ролью `monitoring.editor`.
2. Создайте API-ключ для сервисного аккаунта.
3. Создайте ресурс `PrometheusRemoteWrite`:

```
d8 k apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <URL_ЗАПИСИ_МЕТРИК>
  bearerToken: <API_КЛЮЧ>
EOF
```

Где:

- `<URL_ЗАПИСИ_МЕТРИК>` — URL со страницы Yandex Monitoring/Prometheus/Запись метрик.
- `<API_КЛЮЧ>` — API-ключ, созданный на предыдущем шаге. Например, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs*****`.

Также вы можете указать дополнительные параметры в соответствии с документацией.

Подробнее с данной функциональностью можно ознакомиться в документации Yandex Cloud.

Чтение метрик через Grafana

1. Создайте сервисный аккаунт с ролью `monitoring.viewer`.
2. Создайте API-ключ для сервисного аккаунта.
3. Создайте ресурс `GrafanaAdditionalDatasource`:

```
d8 k apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <URL_ЧТЕНИЕ_МЕТРИК_ЧЕРЕЗ_GRAFANA>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_КЛЮЧ>
EOF
```

Где:

- `<URL_ЧТЕНИЕ_МЕТРИК_ЧЕРЕЗ_GRAFANA>` — URL со страницы Yandex Monitoring/Prometheus/Чтение метрик через Grafana.
- `<API_КЛЮЧ>` — API-ключ, созданный на предыдущем шаге. Например, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs*****`.

Также вы можете указать дополнительные параметры в соответствии с документацией.

Подробнее с данной функциональностью можно ознакомиться в документации Yandex Cloud.

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

YandexInstanceClass

Scope: Cluster

- `v1`
- `v1alpha1`

Параметры группы Yandex Instance'ов (виртуальных машин), которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `NodeGroup` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalLabels**
объект

Дополнительные лейблы, которые будут присвоены созданным виртуальным машинам.

Пример:

```
additionalLabels:
  project: cms-production
  severity: critical
```

- **spec.additionalSubnets**
массив строк

Список дополнительных подсетей, которые будут подключены к виртуальной машине.

Пример:

```
additionalSubnets:
- b0csh41c1or82vuch89v
- e2lgddi5svochh5fbq96
```

- **spec.assignPublicIPAddress**
булевый

Необходимость присвоения публичных IP-адресов виртуальным машинам.

По умолчанию: `false`

Пример:

```
assignPublicIPAddress: false
```

- **spec.coreFraction**
целочисленный

Базовый уровень производительности каждого ядра CPU у создаваемых виртуальных машин. Подробнее об уровнях производительности.

По умолчанию: `100`

Допустимые значения: `5`, `20`, `50`, `100`

Пример:

```
coreFraction: 20
```

- **spec.cores**
целочисленный

Обязательный параметр

Количество ядер у создаваемых виртуальных машин.

Пример:

```
cores: 4
```

- **spec.diskSizeGB**
целочисленный

Размер диска у виртуальных машин. Значение указывается в `Гиб`.

По умолчанию: `50`

Пример:

```
diskSizeGB: 20
```

- **spec.diskType**
строка

Тип диска у виртуальных машин.

Размер дисков `network-ssd-nonreplicated` и `network-ssd-io-m3` должен быть кратен 93 GB.

Подробнее о возможных типах дисков можно узнать в документации провайдера.

По умолчанию: `network-hdd`

Допустимые значения: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`, `network-ssd-io-m3`

Пример:

```
diskType: network-hdd
```

- **spec.gpus**

целочисленный

Количество графических адаптеров у создаваемых виртуальных машин.

По умолчанию: `0`

Пример:

```
gpus: 4
```

- **spec.imageID**

строка

Идентификатор образа, который будет установлен в заказанные виртуальные машины.

По умолчанию используется образ, указанный в параметре `masterNodeGroup.instanceClass.imageID`.

Пример:

```
imageID: fd83ica41cade1mj35sr
```

- **spec.mainSubnet**

строка

Имя основной сети (ID), к которой будет подключен основной сетевой интерфейс виртуальной машины.

Если параметр не задан, то основная сеть определяется автоматически согласно следующей логике: если задан список сетей в параметре `existingZoneToSubnetIDMap`, то сеть определяется из указанного списка, иначе используется созданная Deckhouse сеть.

Пример:

```
mainSubnet: e9bnc7g9mu9mper9clk4
```

- **spec.memory**

целочисленный

Обязательный параметр

Количество оперативной памяти (в мегабайтах) у создаваемых виртуальных машин.

Пример:

```
memory: 8192
```

- **spec.networkType**

строка

Тип сети: обычная или программно-ускоренная.

i В кластерах, использующих Machine Controller Manager (MCM), изменение параметра `networkType` не приводит к автоматическому пересозданию существующих `CloudEphemeral`-узлов. Подробности — в FAQ.

По умолчанию: `Standard`

Допустимые значения: `Standard`, `SoftwareAccelerated`

- **spec.platformID**

строка

ID платформы. Список существующих платформ.

По умолчанию: `standard-v3`

- **spec.preemptible**

булевый

Необходимость заказа прерываемых виртуальных машин (`preemptible`-инстансов).

Подробнее о прерываемых виртуальных машинах можно узнать в документации провайдера.

По умолчанию: `false`

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

Параметры группы Yandex Instance'ов (виртуальных машин), которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `NodeGroup` модуля `node-manager`.

- **spec**
объект

Обязательный параметр

- **spec.additionalLabels**
объект

Дополнительные лейблы, которые будут присвоены созданным виртуальным машинам.

Пример:

```
additionalLabels:
  project: cms-production
  severity: critical
```

- **spec.additionalSubnets**
массив строк

Список дополнительных подсетей, которые будут подключены к виртуальной машине.

Пример:

```
additionalSubnets:
- b0csh41c1or82vuch89v
- e2lgddi5svochh5fbq96
```

- **spec.assignPublicIPAddress**
булевый

Необходимость присвоения публичных IP-адресов виртуальным машинам.

По умолчанию: `false`

Пример:

```
assignPublicIPAddress: false
```

- **spec.coreFraction**
целочисленный

Базовый уровень производительности каждого ядра CPU у создаваемых виртуальных машин. Подробнее об уровнях производительности.

По умолчанию: `100`

Допустимые значения: `5`, `20`, `50`, `100`

Пример:

```
coreFraction: 20
```

- **spec.cores**
целочисленный

Обязательный параметр

Количество ядер у создаваемых виртуальных машин.

Пример:

```
cores: 4
```

- **spec.diskSizeGB**
целочисленный

Размер диска у виртуальных машин. Значение указывается в `Гиб`.

По умолчанию: `20`

Пример:

```
diskSizeGB: 50
```

- **spec.diskType**

строка

Тип диска у виртуальных машин.

Размер дисков `network-ssd-nonreplicated` и `network-ssd-io-m3` должен быть кратен 93 GB.

Подробнее о возможных типах дисков можно узнать в документации провайдера.

По умолчанию: `network-hdd`

Допустимые значения: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`

Пример:

```
diskType: network-hdd
```

- **spec.gpus**

целочисленный

Количество графических адаптеров у создаваемых виртуальных машин.

По умолчанию: `0`

Пример:

```
gpus: 4
```

- **spec.imageID**

строка

Идентификатор образа, который будет установлен в заказанные виртуальные машины.

По умолчанию используется образ, указанный в параметре `masterNodeGroup.instanceClass.imageID`.

Пример:

```
imageID: fd83ica41cade1mj35sr
```

- **spec.mainSubnet**

строка

Имя основной сети (ID), к которой будет подключен основной сетевой интерфейс виртуальной машины.

Если параметр не задан, то основная сеть определяется автоматически согласно следующей логике: если задан список сетей в параметре `existingZoneToSubnetIDMap`, то сеть определяется из указанного списка, иначе используется созданная Deckhouse сеть.

Пример:

```
mainSubnet: e9bnc7g9mu9mper9clk4
```

- **spec.memory**

целочисленный

Обязательный параметр

Количество оперативной памяти (в мегабайтах) у создаваемых виртуальных машин.

Пример:

```
memory: 8192
```

- **spec.networkType**

строка

Тип сети: `STANDARD` или `SOFTWARE_ACCELERATED`.

По умолчанию: `STANDARD`

Допустимые значения: `STANDARD`, `SOFTWARE_ACCELERATED`

- **spec.platformID**

строка

ID платформы. Список существующих платформ.

По умолчанию: `standard-v2`

- **spec.preemptible**

булевый

Необходимость заказа прерываемых виртуальных машин (preemptible-инстансов).

Подробнее о прерываемых виртуальных машинах можно узнать в документации провайдера.

По умолчанию: `false`

FAQ

Настройка INTERNAL LoadBalancer

Для настройки INTERNAL LoadBalancer'a установите аннотацию для сервиса:

```
yandex.cpi.flant.com/listener-subnet-id: SubnetID
```

Аннотация указывает, какой subnet будет слушать LoadBalancer.

Использование отдельной целевой группы для NodeGroup

По умолчанию Yandex Cloud Controller Manager добавляет все подходящие узлы кластера в целевую группу по умолчанию. Чтобы назначить узлам отдельную целевую группу, используйте аннотацию `yandex.cpi.flant.com/target-group-name-prefix`. Описание аннотации приведено в разделе «Аннотации объекта Service».

1. В NodeGroup, узлы которой должны входить в отдельную целевую группу, задайте аннотацию `yandex.cpi.flant.com/target-group-name-prefix` в параметре `spec.nodeTemplate.annotations`. Например:

```
spec:
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
```

Аннотация распространяется на объекты Node. Yandex Cloud Controller Manager добавляет в целевую группу все подходящие узлы с таким значением аннотации независимо от их NodeGroup.

2. В объекте Service типа LoadBalancer задайте ту же аннотацию с тем же значением. Например:

```
metadata:
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
```

Аннотация на Service определяет целевую группу, которую использует LoadBalancer, но не выбирает узлы.

3. Убедитесь, что значения `yandex.cpi.flant.com/target-group-name-prefix` в NodeGroup и Service совпадают. В примере выше в обоих ресурсах используется значение `frontend-`.
4. Целевая группа будет создана с именем, сформированным по шаблону:

```
<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
```

Например, если значение аннотации — `frontend-`, имя кластера — `my-cluster-`, а идентификатор сети — `enp123456789`, имя целевой группы будет следующим:

```
frontend-my-cluster-enp123456789
```

Пример конфигурации:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
  ---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer

```

⚠ Yandex Cloud не позволяет одному целевому ресурсу (target), определяемому парой (SubnetID, IP), одновременно находиться в нескольких целевых группах.

При изменении целевой группы Yandex Cloud Controller Manager сначала удаляет ресурс из текущей группы, а затем добавляет в новую. Во время перемещения возможен кратковременный перерыв в обработке трафика.

Узлы, для которых задан отдельный префикс целевой группы, исключаются из целевой группы по умолчанию. Поэтому балансировщики, использующие её, перестают направлять трафик на такие узлы.

Такое автоматическое перемещение выполняется только между целевыми группами, которыми управляет Yandex Cloud Controller Manager. Если ресурс уже находится в другой целевой группе, не управляемой Yandex Cloud Controller Manager, удалите ресурс из неё перед использованием аннотации.

После изменения или удаления префикса прежняя целевая группа может остаться пустой. Если она больше не используется балансировщиками, удалите её вручную.

Резервирование публичного IP-адреса

Для использования в `externalIPAddresses` и `natInstanceExternalAddress` (также может быть использован для bastion-хоста) выполните следующую команду:

```

$ yc vpc address create --external-ipv4 zone=ru-central1-a
id: e9b4cfmmnc1mhgij75n7
folder_id: b1gog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
  requirements: {}
reserved: true

```

Проблемы `dhcpOptions` и пути их решения

Использование в настройках DHCP-серверов адресов DNS, отличающихся от предоставляемых Yandex Cloud, является временным решением. От него можно будет отказаться, когда Yandex Cloud введет услугу Managed DNS. Чтобы обойти ограничения, описанные ниже, рекомендуется использовать `stubZones` из модуля `kube-dns`.

Изменение параметров

Обратите внимание на следующие особенности:

1. При изменении данных параметров требуется выполнить `netplan apply` или аналог, форсирующий обновление DHCP lease.
2. Потребуется перезапуск всех подов `hostNetwork` (особенно `kube-dns`), чтобы пересчитать новый `resolv.conf`.

Особенности использования

При использовании опции `dhcpOptions` все DNS-запросы начнут идти через указанные DNS-серверы. Эти DNS-серверы **должны** разрешать внешние DNS-имена, а также при необходимости разрешать DNS-имена внутренних ресурсов.

Не используйте эту опцию, если указанные рекурсивные DNS-серверы не могут разрешать тот же список зон,

что сможет разрешать рекурсивный DNS-сервер в подсети Yandex Cloud.

Изменение `networkType` не пересоздаёт `CloudEphemeral`-узлы

В кластерах, использующих Machine Controller Manager (MCM), изменение параметра `networkType` в ресурсе `YandexInstanceClass` не приводит к автоматическому пересозданию существующих `CloudEphemeral`-узлов. Хотя `MachineClass` обновляется, тип сетевого ускорения у уже созданных виртуальных машин в Yandex Cloud не изменяется.

DKP намеренно не учитывает параметр `networkType` при определении необходимости пересоздания `CloudEphemeral`-узлов в MCM. Если бы он учитывался, обновление DKP привело бы к пересозданию `CloudEphemeral`-узлов во всех кластерах, где `networkType` уже указан, даже если его значение не изменялось.

В Cluster API параметр `networkType` учитывается при определении необходимости пересоздания узлов, поэтому его изменение запускает их обновление автоматически. Новые `CloudEphemeral NodeGroup` в Yandex по умолчанию используют CAPI.

Если вы изменили `networkType` в кластере на MCM и хотите, чтобы новое значение применилось к уже существующим `CloudEphemeral`-узлам, пересоздайте их вручную — инструкция в документации.

Назначение произвольного `StorageClass` используемого по умолчанию

Чтобы назначить произвольный `StorageClass` используемым по умолчанию для ваших PVC, укажите его имя в параметре `defaultClusterStorageClass` модуля `global`. Обратите внимание, что после этого аннотация `storageclass.kubernetes.io/is-default-class='true'` снимется со `StorageClass`'а, который ранее был указан как используемый по умолчанию.

```
d8 k edit mc global
```

Изменение размера PVC

Размер существующего PVC можно увеличить путем изменения значения параметра `spec.resources.requests.storage` без остановки и пересоздания использующего его пода.

После изменения значения `spec.resources.requests.storage` CSI-драйвер последовательно:

- увеличивает размер диска в Yandex Cloud;
- обновляет размер связанного `PersistentVolume`;
- выполняет расширение файловой системы на узле, к которому подключён том.

Во время операции под продолжает работать, а смонтированный том остаётся доступным приложению. После завершения увеличения новый размер файловой системы становится доступен внутри контейнера без перезапуска пода.



Уменьшение размера PVC не поддерживается.

Чтобы увеличить PVC, выполните следующие действия:

1. Получите имя `StorageClass`, используемого PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC> \
-o jsonpath='{.spec.storageClassName}{"\n"}'
```

Где:

- `<НЕЙМСПЕЙС>` — неймспейс, в котором находится PVC;
- `<ИМЯ_PVC>` — имя PVC, размер которого необходимо увеличить.

Например:

```
d8 k -n production get pvc application-data \
-o jsonpath='{.spec.storageClassName}{"\n"}'
```

Пример вывода команды:

```
network-ssd
```

В этом примере PVC `application-data` использует `StorageClass` `network-ssd`.

2. Убедитесь, что `StorageClass` разрешает увеличение томов:

```
d8 k get storageclass <ИМЯ_STORAGECLASS> \
-o jsonpath='{.allowVolumeExpansion}{"\n"}'
```

Где `<ИМЯ_STORAGECLASS>` — имя `StorageClass`, полученное на предыдущем шаге.

Например:

```
d8 k get storageclass network-ssd \
-o jsonpath='{.allowVolumeExpansion}{"\n"}'
```

Пример вывода команды:

```
true
```

3. Проверьте текущее состояние и размер PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production get pvc application-data
```

Пример вывода:

NAME	STATUS	VOLUME	CAPACITY	ACCESS
MODES STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	20Gi	RW0
network-ssd				

Убедитесь, что:

- PVC находится в состоянии `Bound`;
- в поле `CAPACITY` указан текущий размер PVC;
- в поле `STORAGECLASS` указан StorageClass, проверенный на предыдущем шаге.

4. Увеличьте размер PVC:

```
d8 k -n <НЕЙМСПЕЙС> edit pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production edit pvc application-data
```

В поле `spec.resources.requests.storage` укажите новый размер PVC:

```
spec:
  resources:
    requests:
      storage: 30Gi
```

В этом примере размер PVC увеличивается до 30Gi.

Сохраните изменения и закройте редактор.



Для StorageClass `network-ssd-nonreplicated` и `network-ssd-io-m3` размер должен быть кратен 93Gi.

5. Дождитесь увеличения PVC:

```
d8 k -n <НЕЙМСПЕЙС> get pvc <ИМЯ_PVC> --watch
```

Где:

- `<НЕЙМСПЕЙС>` — неймспейс, в котором находится PVC;
- `<ИМЯ_PVC>` — имя PVC, размер которого увеличивается.

Например:

```
d8 k -n production get pvc application-data --watch
```

Во время увеличения в поле `CAPACITY` может отображаться прежний размер:

NAME	STATUS	VOLUME	CAPACITY	ACCESS
MODES STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	20Gi	RWO
network-ssd				

Операция завершена, когда в поле `CAPACITY` отображается новый размер PVC:

NAME	STATUS	VOLUME	CAPACITY	ACCESS
MODES STORAGECLASS				
application-data	Bound	pvc-65e92674-077c-4b4f-b65d-19e92f04e103	30Gi	RWO
network-ssd				

6. Проверьте события PVC:

```
d8 k -n <НЕЙМСПЕЙС> describe pvc <ИМЯ_PVC>
```

Например:

```
d8 k -n production describe pvc application-data
```

Во время увеличения могут появиться следующие события:

```
ExternalExpanding
Resizing
FileSystemResizeRequired
```

Об успешном увеличении файловой системы свидетельствует событие:

```
FileSystemResizeSuccessful
```

Например:

```
Normal FileSystemResizeSuccessful kubelet MountVolume.NodeExpandVolume succeeded for
volume "pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
```

7. Проверьте размер файловой системы внутри пода:

```
d8 k -n <НЕЙМСПЕЙС> exec <ИМЯ_ПОДА> -- \
df -hT <ТОЧКА_МОНТИРОВАНИЯ>
```

Где:

- `<НЕЙМСПЕЙС>` — неймспейс, в котором находится под;
- `<ИМЯ_ПОДА>` — имя пода, использующего PVC;
- `<ТОЧКА_МОНТИРОВАНИЯ>` — путь внутри контейнера, в который смонтирован PVC.

Например:

```
d8 k -n production exec application-0 -- \
df -hT /data
```

Пример вывода:

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
/dev/vde	ext4	29.4G	22M	29.4G	1%	/data

i Фактический размер файловой системы может быть немного меньше размера PVC из-за служебных данных файловой системы.

Добавление CloudStatic-узлов в кластер

В метаданные виртуальных машин, которые вы хотите включить в кластер в качестве узлов, добавьте (Изменить VM -> Метаданные) ключ `node-network-cidr` со значением `nodeNetworkCIDR` для кластера.

`nodeNetworkCIDR` кластера можно узнать, воспользовавшись следующей командой:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration -o json | jq --raw-output '.data."cloud-provider-cluster-configuration.yaml"' | base64 -d | grep '^nodeNetworkCIDR'
```

Создание кластера в новом VPC и разворачивание bastion-хост для доступа к узлам

1. Выполните bootstrap базовой инфраструктуры кластера:

```
dhctl bootstrap-phase base-infra --config config.yml
```

2. Создайте bastion-хост:

```
yc compute instance create \
--name bastion \
--hostname bastion \
--create-boot-disk image-family=ubuntu-2404-lts,image-folder-id=standard-images,size=20,type=network-hdd \
--memory 2 \
--cores 2 \
--core-fraction 100 \
--ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
--zone ru-central1-a \
--public-address 178.154.226.159
```

i Замените здесь `<SSH_PUBLIC_KEY_FILE>` на имя вашего публичного ключа. Например, для ключа с RSA-шифрованием это будет `id_rsa.pub`, а для ключа с ED25519-шифрованием `id_ed25519.pub`.

3. Продолжите установку кластера, указав данные bastion-хоста. На вопрос про кеш Terraform ответьте `y`:

```
dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
--ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --config=/config.yml
```

i Замените здесь `<SSH_PRIVATE_KEY_FILE>` на имя вашего приватного ключа. Например, для ключа с RSA-шифрованием это может быть `id_rsa`, а для ключа с ED25519-шифрованием — `id_ed25519`.

Длительное переключение на заказ узлов в менее приоритетных группах

Если переключение на заказ узлов в менее приоритетных группах занимает много времени (например, когда для групп узлов с preemptible-инстансами установлен наивысший приоритет и при недоступности таких инстансов заказ узлов из других групп происходит очень долго), воспользуйтесь инструкцией.

Примеры

Ниже представлен пример конфигурации cloud-провайдера Yandex Cloud.

Пример конфигурации модуля

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    additionalExternalNetworkIDs:
      - enp6t4snovl2ko4p15em
```

Пример custom resource YandexInstanceClass

```
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: test
spec:
  cores: 4
  memory: 8192
```

LoadBalancer

Аннотации объекта Service

В кластере заданы значения по умолчанию для размещения ресурсов балансировщиков (сеть для целевой группы и подсеть для Listener). Эти значения выставляются автоматически во время развёртывания кластера и могут быть переопределены аннотациями на уровне конкретного Service.

i Поведение по умолчанию (внешний или внутренний LB) зависит от конфигурации кластера. Для явного выбора типа используйте аннотацию `yandex.cpi.flant.com/loadbalancer-external`.

В Yandex Cloud Controller Manager поддерживаются следующие аннотации:

1. `yandex.cpi.flant.com/target-group-network-id` — указывает NetworkID, в котором будет создана целевая группа для данного Service. Переопределяет соответствующее значение по умолчанию.
2. `yandex.cpi.flant.com/listener-subnet-id` — задаёт SubnetID для Listener'ов создаваемого LB для данного Service. Переопределяет соответствующее значение по умолчанию.
3. `yandex.cpi.flant.com/listener-address-ipv4` — задаёт предопределённый IPv4-адрес для Listener'ов (поддерживаются и внутренние, и внешние LB).
4. `yandex.cpi.flant.com/loadbalancer-external` — включает создание внешнего (external) LB для данного Service (используйте, если нужно явно создать внешний балансировщик). Переопределяет поведение по умолчанию.
5. `yandex.cpi.flant.com/target-group-name-prefix` — задаёт префикс имени целевой группы, которую использует LoadBalancer. Аннотация на Service определяет целевую группу, но не выбирает узлы. Чтобы включить узлы в целевую группу, задайте аннотацию с тем же значением в параметре `spec.nodeTemplate.annotations` ресурса NodeGroup. Yandex Cloud Controller Manager включает в целевую группу все подходящие узлы с таким значением аннотации независимо от их NodeGroup. Имя целевой группы формируется как `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`. Подробнее — в разделе «Использование отдельной целевой группы для NodeGroup».

Если для control plane или master-узлов создаются отдельные целевые группы, добавьте на master-узлы лейбл `node.kubernetes.io/exclude-from-external-load-balancers: ""`. Это предотвратит попытки контроллера автоматически добавлять master-узлы в новые целевые группы для балансировщиков.

Если вы создаёте собственный балансировщик для master-узлов и хотите, чтобы YCC также мог размещать свои балансировщики на master-узлах, заранее создайте целевую группу с именем по маске `${CLUSTER-NAME}-${VPC.ID}`.

Проверки состояния целевой группы

Параметры healthcheck'ов (для создаваемых целевых групп балансировщика):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — как часто запускать проверку, в секундах (по умолчанию 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — сколько ждать ответа от эндпоинта, в секундах. Если за это время ответ не получен, проверка считается неуспешной (по умолчанию 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — сколько подряд неуспешных проверок нужно, чтобы пометить эндпоинт как неработоспособный (unhealthy) и исключить его из балансировки (по умолчанию 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — сколько подряд успешных проверок нужно, чтобы вернуть эндпоинт в статус работоспособный (healthy) и снова включить его в балансировку (по умолчанию 2).

Схемы размещения

Поддерживаются три схемы размещения. Ниже подробнее о каждой из них.

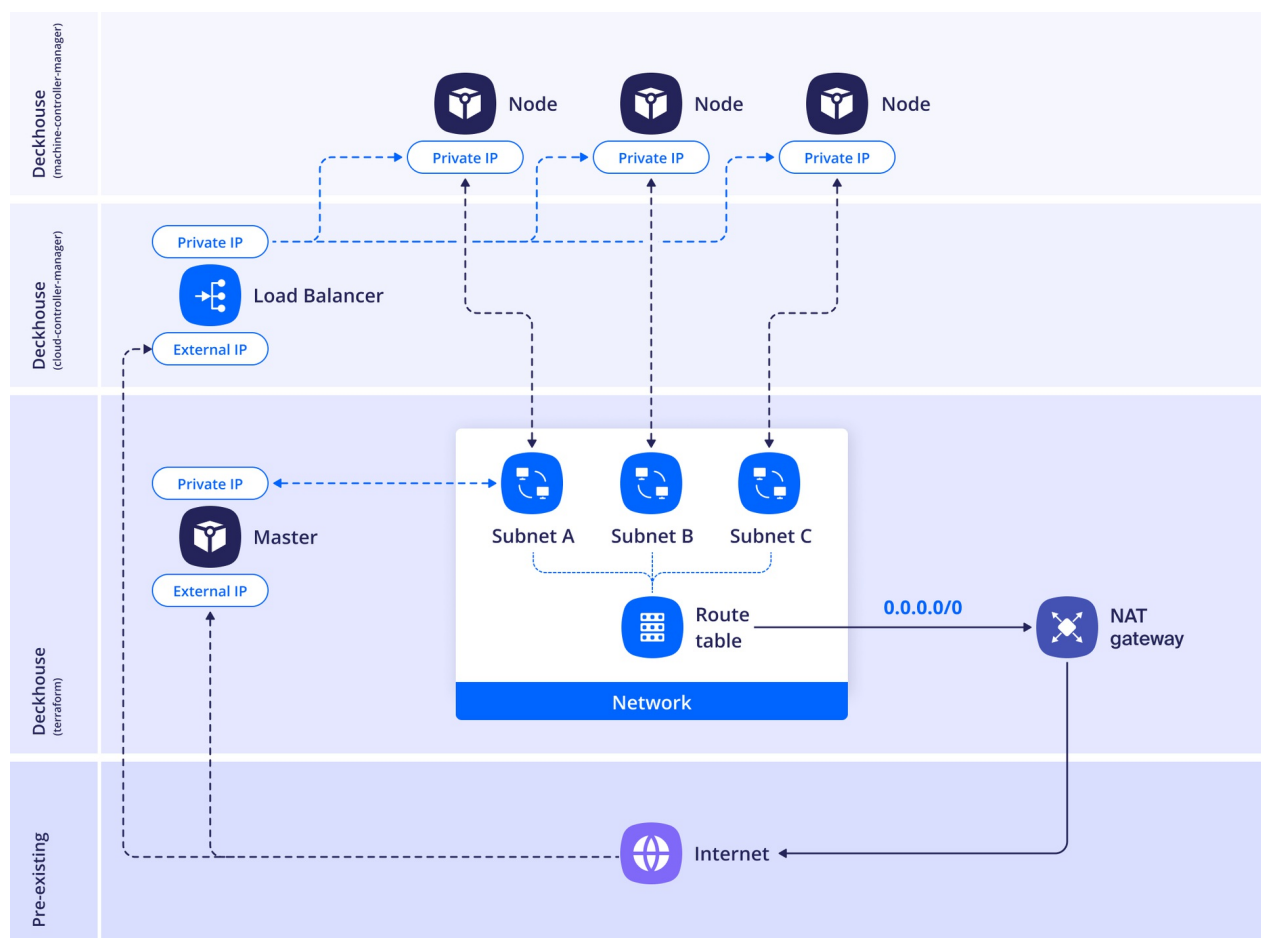
Standard



В данной схеме размещения узлы не будут иметь публичных IP-адресов и будут выходить в интернет через NAT-шлюз (NAT Gateway) Yandex Cloud. NAT-шлюз (NAT Gateway) использует случайные публичные IP-адреса из выделенных диапазонов. Из-за этого невозможно добавить в белый список (whitelist) адреса облачных ресурсов, находящихся за конкретным NAT-шлюзом, на стороне других сервисов.



Так как в данной схеме размещения узлы создаются без публичных IP-адресов, для установки кластера на master-узел должен быть SSH-доступ с машины, на которой запускается dhctl: напрямую по приватной сети или через bastion-хост. Если master-узел недоступен напрямую, запускайте установку с параметрами `--ssh-bastion-host`, `--ssh-bastion-user` и, при необходимости, `--ssh-bastion-port`.



Пример конфигурации схемы размещения:

```

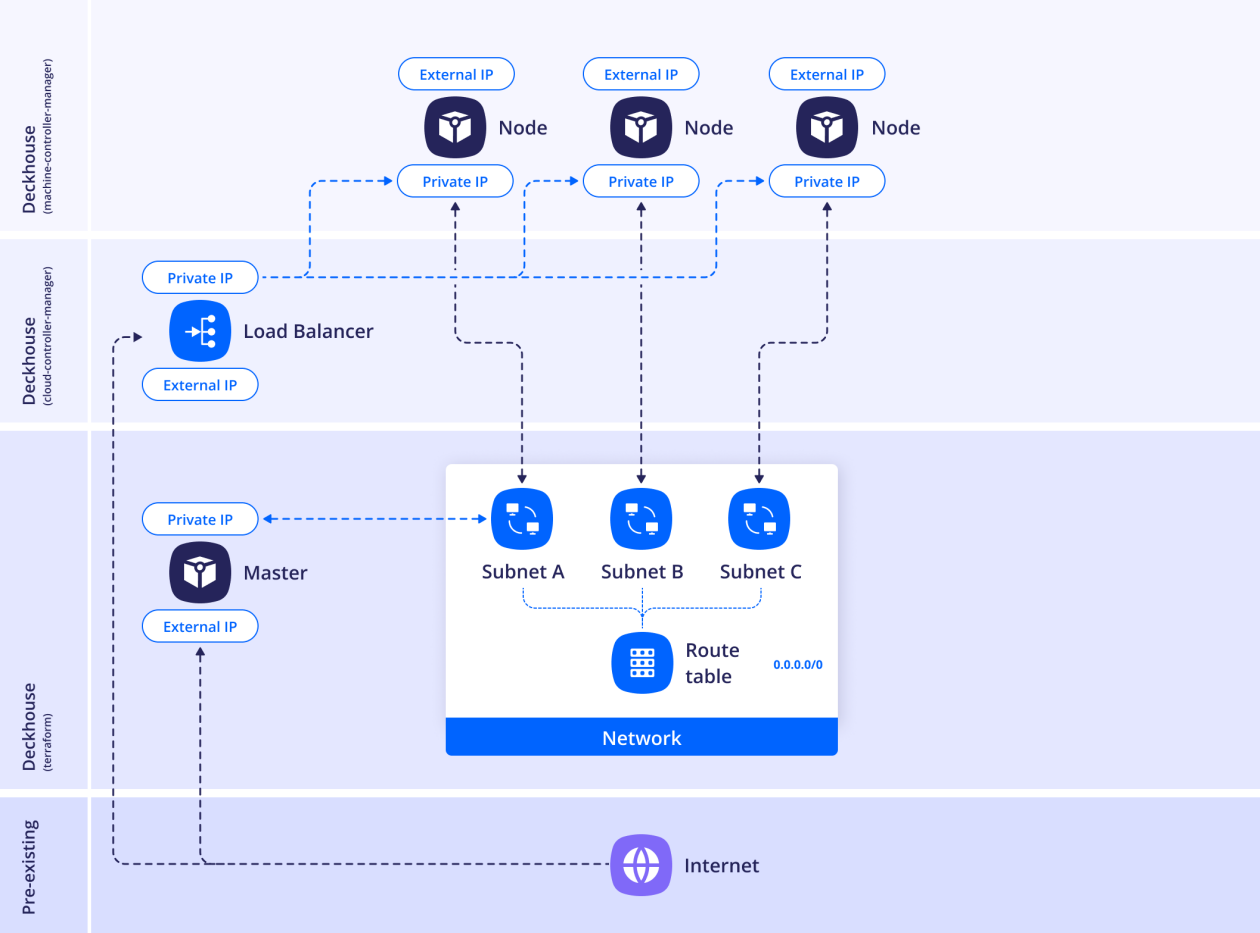
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  zones:
    - ru-central1-a
    - ru-central1-b
    - ru-central1-d
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    # Опционально: раскомментируйте, чтобы назначить master-узлам публичные IP-адреса.
    # externalIPAddresses:
    # - "<ZONE_A_EXTERNAL_IP_MASTER_1>"
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    # - <ZONE_D_SUBNET_ID>
    additionalLabels:
      takes: priority
nodeGroups:
- name: worker
  replicas: 2
  zones:
    - ru-central1-a
    - ru-central1-b
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    # Опционально: раскомментируйте, чтобы назначить worker-узлам публичные IP-адреса.
    # externalIPAddresses:
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    additionalLabels:
      role: example
  labels:
    billing: prod
dhcpOptions:
  domainName: test.local
  domainNameServers:
    - <DNS_SERVER_1>
    - <DNS_SERVER_2>

```

WithoutNAT

В данной схеме размещения NAT (любого вида) не используется, а каждому узлу выдается публичный IP-адрес.

Внимание. В модуле `cloud-provider-yandex` нет поддержки групп безопасности (security group), поэтому все узлы кластера будут доступны без ограничения подключения.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    externalIPAddresses:
      - "Auto"
      - "Auto"
      - "Auto"
    externalSubnetIDs:
      - <ZONE_A_SUBNET_ID>
      - <ZONE_B_SUBNET_ID>
      - <ZONE_D_SUBNET_ID>
    zones:
      - ru-central1-a
      - ru-central1-b
      - ru-central1-d
nodeGroups:
- name: worker
  replicas: 2
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    externalIPAddresses:
      - "<ZONE_A_EXTERNAL_IP_WORKER_1>"
      - "Auto"
    externalSubnetIDs:
      - <ZONE_A_SUBNET_ID>
      - <ZONE_B_SUBNET_ID>
    zones:
      - ru-central1-a
      - ru-central1-b
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
    - <DNS_SERVER_1>
    - <DNS_SERVER_2>

```

WithNATInstance

В данной схеме размещения в отдельной подсети создается NAT-инстанс, а в таблицу маршрутизации подсетей зон добавляется правило с маршрутом на 0.0.0.0/0 с NAT-инстансом в качестве nexthop'a. Подсеть выделяется для предотвращения петли маршрутизации и не должна пересекаться с другими сетями, используемыми в кластере.

Для размещения NAT-инстанса в существующей подсети используйте параметр `withNATInstance.internalSubnetID` — инстанс будет создан в зоне, соответствующей этой подсети.

Если необходимо создать новую подсеть, укажите параметр `withNATInstance.internalSubnetCIDR` — в ней будет размещён NAT-инстанс.

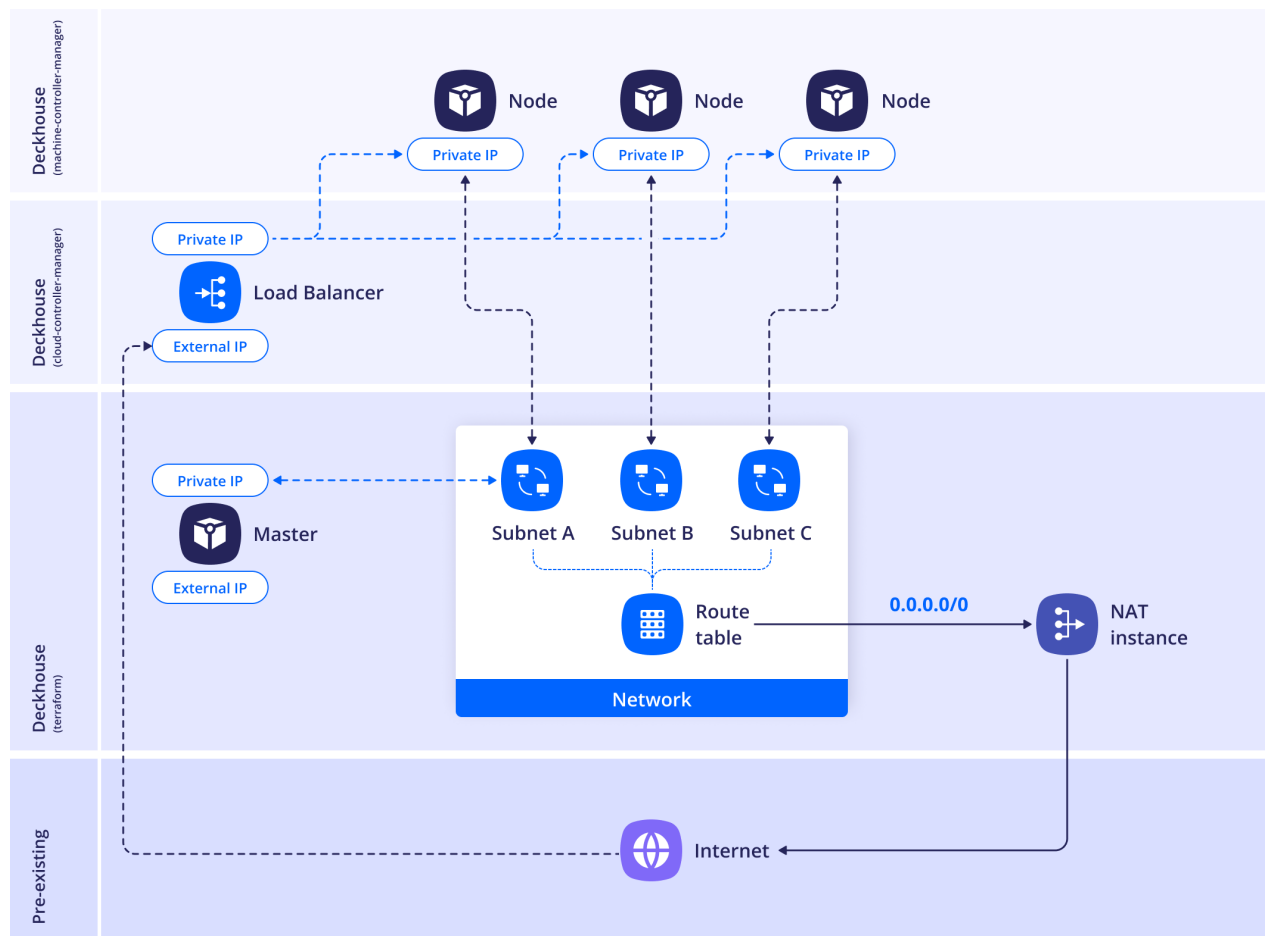
Обязателен один из параметров: `withNATInstance.internalSubnetID` или

`withNATInstance.internalSubnetCIDR`.

При указании параметра `withNATInstance.internalSubnetCIDR` NAT-инстанс будет создан в единственном экземпляре в зоне `a`.

При указании параметра `withNATInstance.internalSubnetID` NAT-инстанс будет создан в единственном экземпляре в зоне, указанной в настройках обозначенной сети.

Если `withNATInstance.externalSubnetID` указан в дополнение к предыдущим, NAT-инстанс будет подключен к нему через вторичный интерфейс.



Пример конфигурации схемы размещения:

```

apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithNATInstance
withNATInstance:
  natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
  internalSubnetID: <INTERNAL_SUBNET_ID>
  externalSubnetID: <EXTERNAL_SUBNET_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
      "id": "id",
      "service_account_id": "service_account_id",
      "key_algorithm": "RSA_2048",
      "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID...AQAB\n-----END PUBLIC KEY-----\n",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE...1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>

```

- i** В примере master и worker-узлы создаются без публичных IP-адресов и без дополнительных внешних сетевых интерфейсов. Исходящий трафик из подсетей кластера проходит через NAT-инстанс.
- NAT-инстанс не используется инсталлятором `dhctl` автоматически в качестве bastion-хоста или jump-хоста. Для установки кластера у master-узла должен быть SSH-доступ с машины, на которой запускается `dhctl`: напрямую или через bastion-хост.
- Если master-узел не имеет публичного IP-адреса или недоступен напрямую, запускайте установку с параметрами `--ssh-bastion-host` и `--ssh-bastion-user`.
- Если необходимо подключаться к master-узлу напрямую по публичному IP-адресу, укажите для него параметры `externalIPAddresses` и `externalSubnetIDs`. Учитывайте, что интерфейс из сети, указанной в `externalSubnetIDs`, будет использоваться как шлюз по умолчанию для узла.

Cloud provider — zVirt

Модуль `cloud-provider-zvirt` обеспечивает интеграцию Deckhouse Kubernetes Platform с zVirt. Он предоставляет возможность модулю `node-manager` использовать ресурсы zVirt при заказе узлов для NodeGroup.

Функции модуля `cloud-provider-zvirt`:

- Управление ресурсами zVirt через `cloud-controller-manager`: актуализирует метаданные виртуальных машин и узлов Kubernetes и удаляет из Kubernetes узлы, которых больше нет в zVirt.
- Заказ дисков через CSI-драйвер zVirt (csi.ovirt.org), чтобы из кластера можно было заказывать PersistentVolume.
- Заказ CloudPermanent-узлов с помощью Terraform/OpenTofu-провайдера `terraform-provider-ovirt`.
- Заказ CloudEphemeral-узлов через Cluster API (CAPI). Параметры виртуальных машин задаются в ресурсе

- ZvirtInstanceClass.
- Регистрация в модуле `node-manager`, чтобы ZvirtInstanceClass можно было указывать при описании NodeGroup.
- Автоматическое включение CNI для новых кластеров. По умолчанию используется `cni-cilium`.

Настройки

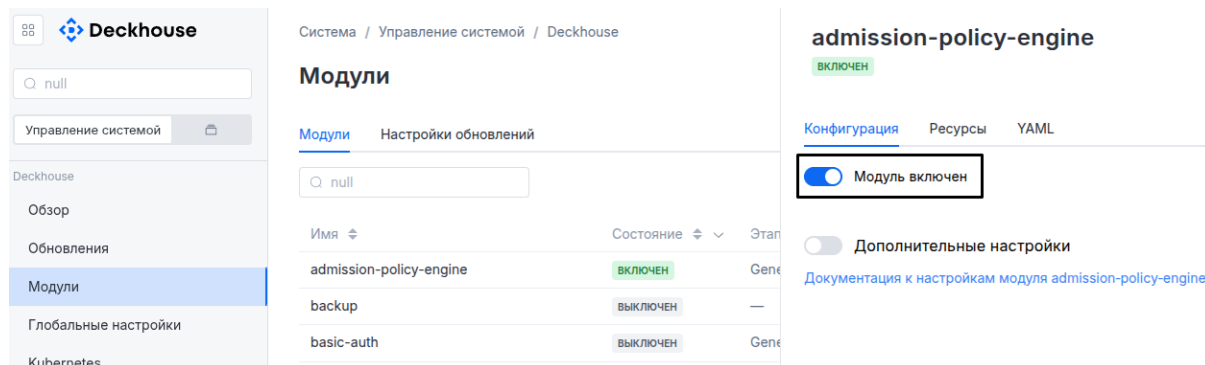
Модуль автоматически включается для всех облачных кластеров, развернутых в zVirt.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-zvirt`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cloud-provider-zvirt`:

```
d8 system module enable cloud-provider-zvirt
```

- **С помощью ModuleConfig `cloud-provider-zvirt`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cloud-provider-zvirt` (создайте его, при необходимости).

Пример манифеста для включения модуля `cloud-provider-zvirt`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-zvirt`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cloud-provider-zvirt` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cloud-provider-zvirt`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cloud-provider-zvirt` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cloud-provider-zvirt
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cloud-provider-zvirt`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cloud-provider-zvirt`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-zvirt
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Модуль не имеет настроек.

Настройки провайдера

i Если control plane кластера размещен на виртуальных машинах или bare-metal-серверах, cloud-провайдер использует настройки модуля `cloud-provider-zvirt` в конфигурации Deckhouse. Иначе, если control plane кластера размещен в облаке, cloud-провайдер использует структуру `ZvirtClusterConfiguration` для настройки.

ZvirtClusterConfiguration

Version: deckhouse.io/v1

Описывает конфигурацию облачного кластера в Zvirt.

Используется облачным провайдером, если управляющий слой (control plane) кластера размещен в облаке.

Выполните следующую команду, чтобы изменить конфигурацию в работающем кластере:

```
d8 system edit provider-cluster-configuration
```


i После изменения параметров узлов необходимо выполнить команду `dhctl converge`, чтобы изменения вступили в силу. Подробнее про изменение конфигурации кластера — в разделе «Добавление и управление облачными узлами».

Пример:

```
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
clusterID: b46372e7-0d52-40c7-9bbf-fda31e187088
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
    storageDomainID: c4bf82a5-b803-40c3-9f6c-b9398378f424
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**

строка

Обязательный параметр

Допустимые значения: `deckhouse.io/v1`

- **clusterID**

строка

Обязательный параметр

ID кластера с общими доменами хранения и ЦП одного типа для создания виртуальных машин.

Шаблон: `^[a-fA-F]{8}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{12}$`

Пример:

```
clusterID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **kind**

строка

Обязательный параметр

Допустимые значения: `ZvirtClusterConfiguration`

- **layout**

строка

Обязательный параметр

Название схемы размещения.

Подробнее о возможных схемах размещения провайдера.

- **masterNodeGroup**

объект

Обязательный параметр

Спецификация для описания NodeGroup master-узлов.

i Чтобы изменения вступили в силу, после изменения параметров секции `masterNodeGroup` выполните команду `dhctl converge`.

◦ **masterNodeGroup.instanceClass**
объект

Обязательный параметр

Частичное содержимое полей `ZvirtInstanceClass`.

■ **masterNodeGroup.instanceClass.customNetworkConfig**
объект

Включает пользовательскую статическую настройку сети для группы узлов (NodeGroup).

Если заполнено, необходимо задать `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` и `networkInterfaceNetmask`.

■ **masterNodeGroup.instanceClass.customNetworkConfig.dnsServers**
строка

Список DNS-серверов.

Пример:

```
dnsServers: 8.8.8.8 8.8.4.4
```

■ **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceAddress**
массив строк

Список IP-адресов, которые нужно назначить интерфейсам (по адресу для каждого узла).

Пример:

```
networkInterfaceAddress:
- 192.168.1.10
- 192.168.1.11
```

■ **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceGateway**
строка

Шлюз по умолчанию.

Пример:

```
networkInterfaceGateway: 192.168.1.1
```

■ **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceName**
строка

Имя сетевого интерфейса, для которого будет применена статическая настройка.

Пример:

```
networkInterfaceName: enp1s0
```

■ **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceNetmask**
строка

Маска подсети для интерфейса.

Пример:

```
networkInterfaceNetmask: 255.255.255.0
```

■ **masterNodeGroup.instanceClass.etcdDiskSizeGb**
целочисленный

Etcd disk size in GiB.

По умолчанию: 10

Пример:

```
etcdDiskSizeGb: 10
```

■ **masterNodeGroup.instanceClass.memory**
целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
memory: 8192
```

- **masterNodeGroup.instanceClass.numCPUs**

целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
numCPUs: 2
```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**

целочисленный

Root disk size in GiB to use in zVirt VirtualMachines.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

- **masterNodeGroup.instanceClass.storageDomainID**

строка

Обязательный параметр

ID домена хранения.

Шаблон: `^[^a-fA-F]{8}\-[^a-fA-F]{4}\-[^a-fA-F]{4}\-[^a-fA-F]{4}\-[^a-fA-F]{12}$`

Пример:

```
storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **masterNodeGroup.instanceClass.template**

строка

Обязательный параметр

Имя шаблона, который будет использован при заказе виртуальных машин.

Пример:

```
template: debian-bookworm
```

- **masterNodeGroup.instanceClass.vnicProfileID**

строка

Обязательный параметр

ID профиля vNIC.

Шаблон: `^[^a-fA-F]{8}\-[^a-fA-F]{4}\-[^a-fA-F]{4}\-[^a-fA-F]{4}\-[^a-fA-F]{12}$`

Пример:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **masterNodeGroup.replicas**

целочисленный

Обязательный параметр

Количество создаваемых master-узлов. Для обеспечения кворума важно, чтобы оно было нечетным.

Допустимые значения: `1 <= X`

- **nodeGroups**

массив объектов

Массив дополнительных NodeGroup для создания статических узлов (например, для выделенных frontend-узлов или шлюзов).

- **nodeGroups.instanceClass**

объект

Обязательный параметр

Частичное содержимое полей ZvirtInstanceClass.

- **nodeGroups.instanceClass.customNetworkConfig**

объект

Включает пользовательскую статическую настройку сети для группы узлов (NodeGroup).

Если заполнено, необходимо задать `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` и `networkInterfaceNetmask`.

- **nodeGroups.instanceClass.customNetworkConfig.dnsServers**

строка

Список DNS-серверов.

Пример:

```
dnsServers: 8.8.8.8 8.8.4.4
```

- **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceAddress**

массив строк

Список IP-адресов, которые нужно назначить интерфейсам (по адресу для каждого узла).

Пример:

```
networkInterfaceAddress:  
- 192.168.1.10  
- 192.168.1.11
```

- **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceGateway**

строка

Шлюз по умолчанию.

Пример:

```
networkInterfaceGateway: 192.168.1.1
```

- **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceName**

строка

Имя сетевого интерфейса, для которого будет применена статическая настройка.

Пример:

```
networkInterfaceName: enp1s0
```

- **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceNetmask**

строка

Маска подсети для интерфейса.

Пример:

```
networkInterfaceNetmask: 255.255.255.0
```

- **nodeGroups.instanceClass.etcdDiskSizeGb**

целочисленный

Etcd disk size in GiB.

По умолчанию: 10

Пример:

```
etcdDiskSizeGb: 10
```

- **nodeGroups.instanceClass.memory**

целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам.

Допустимые значения: 1 <= X

Пример:

```
memory: 8192
```

- **nodeGroups.instanceClass.numCPUs**

целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам.

Допустимые значения: `1 <= X`

Пример:

```
numCPUs: 2
```

- **nodeGroups.instanceClass.rootDiskSizeGb**

целочисленный

Root disk size in GiB to use in zVirt VirtualMachines.

По умолчанию: `50`

Пример:

```
rootDiskSizeGb: 50
```

- **nodeGroups.instanceClass.storageDomainID**

строка

ID домена хранения.

Шаблон: `^[a-fA-F]{8}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{12}$`

Пример:

```
storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **nodeGroups.instanceClass.template**

строка

Обязательный параметр

Имя шаблона, который будет использован при заказе виртуальных машин.

Пример:

```
template: debian-bookworm
```

- **nodeGroups.instanceClass.vnicProfileID**

строка

Обязательный параметр

ID профиля vNIC.

Шаблон: `^[a-fA-F]{8}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{4}\-[a-fA-F]{12}$`

Пример:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **nodeGroups.name**

строка

Обязательный параметр

Имя NodeGroup, которое будет использоваться для генерации имен узлов.

- **nodeGroups.nodeTemplate**

Настройки Node-объектов в Kubernetes, которые будут добавлены после регистрации узлов.

- **nodeGroups.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **nodeGroups.nodeTemplate.labels**

объект

Список лейблов, которые будут прикреплены ко всем ресурсам кластера (если они это поддерживают).

Аналогично стандартному полю `metadata.labels`.

Если поменять лейблы в рабочем кластере, после применения изменений необходимо пересоздать все машины.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `values`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**

строка

- **nodeGroups.nodeTemplate.taints.value**

строка

- **nodeGroups.replicas**

целочисленный

Обязательный параметр

Количество узлов.

- **provider**

объект

Обязательный параметр

Параметры для подключения к Zvirt.

- **provider.caBundle**

строка

CA сертификат закодированный base64.

По умолчанию:

- **provider.insecure**

булевый

Установите значение `true`, если Zvirt имеет самоподписанный сертификат.

По умолчанию: `false`

- **provider.password**

строка

Обязательный параметр

Пароль.

- **provider.server**

строка

Обязательный параметр

Хост или IP-адрес zvirt сервера.

- **provider.username**

строка

Обязательный параметр

Логин.

- **sshPublicKey**
строка

Обязательный параметр

Публичный ключ для доступа на узлы.

Подготовка окружения.

Требования к окружению

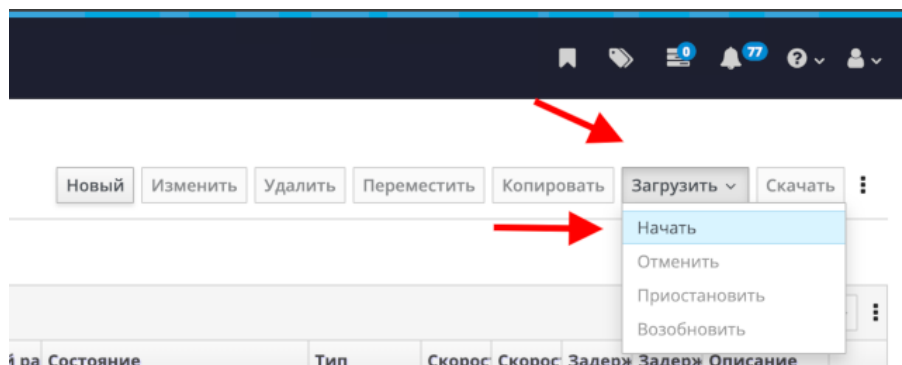
- Рабочая инсталляция zVirt с доступным API;
- Версия zVirt 4.0 - 4.4 ;
- Пользователь с правами доступа к API и управлению виртуальными машинами;
- Хранилище с загруженным cloud-образом операционной системы;
- Подготовленный шаблон виртуальной машины на основе cloud-образа.

Подготовка образа операционной системы

Вендоры операционных систем обычно предоставляют специальные cloud-сборки своих операционных систем для использования в среде виртуализации. Эти сборки, как правило, содержат драйверы для виртуального оборудования, cloud-init, гостевые агенты систем виртуализации и распространяются в виде образов дисков в форматах IMG или QCOW2. Мы рекомендуем использовать такие cloud-образы в качестве ОС на узлах ваших кластеров.

Cloud-образ операционной системы должен быть размещен в хранилище дисков zVirt. Выполните следующие шаги для загрузки образа ОС в хранилище:

1. Перейдите в портал администрирования и откройте раздел *zVirt -> Хранилище -> Диски*.
2. Загрузите cloud-образ ОС в хранилище.



Загрузить образ

Выберите файл

alse-vanilla-1.7.5-cloud-base-mg12.1.2.qcow2

Формат:

QCOW2

Содержимое:

Данные

Размер:

< 1 GiB

QCOW2 Compat:

1.1

Виртуальный размер:

2 GiB

Файл резервной копии:

Нет

Настройки диска

Размер (GiB)

3

Имя

Astra-Linux-Special-Edition-1.7.5

Описание

Astra Linux cloud image

Центр данных

Default

Домен хранения

Local (1639 GiB свободно из 1687 GiB)

Профиль диска

Local

Хост

zvirt.local

Очистить после удаления

Может быть общим

Включить инкрементное резервное копирование

Тест соединения

✓

Тест соединения с ovirt-imageio-проху прошел успешно.

OK

Заккрыть

3. Дождитесь окончания загрузки образа в хранилище. В столбце “Состояние” должен появиться статус “OK”. На этом подготовка образа ОС завершена.

Astra-Linux-Special-Edition-1.7.5	70774afc-b4db-466e-ac...		Local	2 GiB	Загружено 132 из 591 MB	Образ	0MB/s	0MB/s				
-----------------------------------	--------------------------	--	-------	-------	-------------------------	-------	-------	-------	--	--	--	--

Подготовка шаблона виртуальной машины

⚠️ Провайдер поддерживает работу только с одним диском в шаблоне виртуальной машины. Убедитесь, что шаблон содержит только один диск.

Для создания шаблона виртуальной машины перейдите в раздел *Ресурсы* -> *Виртуальные машины* портала администрирования zVirt и создайте новую виртуальную машину со следующими параметрами:

- Раздел *Общие*:
 - Шаблон:** Blank
 - Операционная система:** в зависимости от ОС в cloud-образе.
 - Профиль нагрузки:** Высокая производительность.
 - Виртуальные диски:** Прикрепить -> Выберите загруженный ранее cloud-образ ОС, установите галочку “Загрузочный”, выберите интерфейс “VirtIO”. Другие диски создавать или прикреплять не нужно.

Прикрепить виртуальные диски

Образ Прямой LUN Блочное устройство

	Имя	Описание	Код	Виртуальный р	Актуальный ра	Домен хранения	Интерфейс	R/O	OS	
☒	Astra-Linux-...		5f884ff9-1e...	2 GiB	< 1 GiB	Local	VirtIO-SCSI	☐	☒	

OK Заккрыть

- Сетевые интерфейсы прикреплять не нужно, Deckhouse создаст и настроит их на виртуальной машине самостоятельно во время установки.

Новая виртуальная машина

Общее >

Система

Запуск инициализации

Консоль

Хост

Высокая доступность

Выделение ресурсов

Параметры загрузки

ГСЧ

Доп. параметры

Значок

Foreman/Satellite

Группы сходства

Кластер: Default
Центр данных: Default

Шаблон: Blank | (0)

Операционная система: Other OS

Тип BIOS: Чипсет Q35 с BIOS

Профиль нагрузки: Высокая производительность

Имя: Deckhouse-Template

Описание:

Комментарий:

VM ID:

☐ Без запоминания состояния ☐ Запустить в режиме приостановки

☐ Защита от удаления ☐ Запечатать

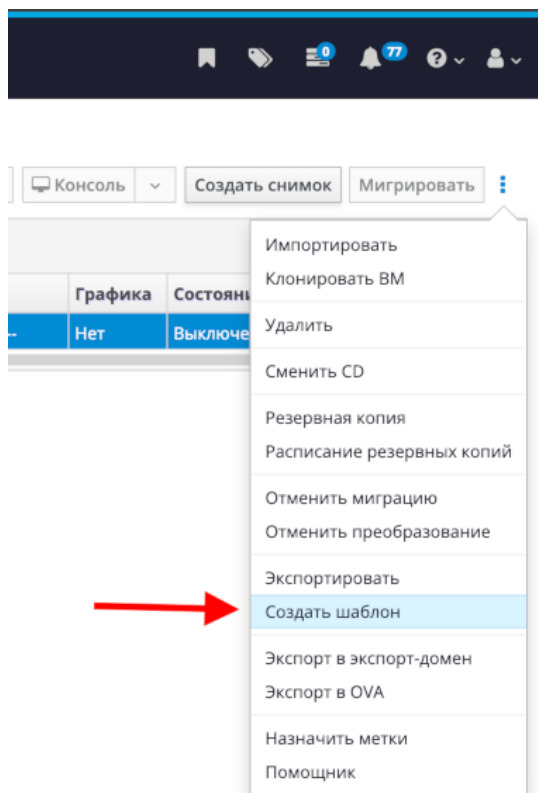
Виртуальные диски
Astra-Linux-Special-Edition-1.7.5: (2 GB) прикрепление (загрузочный) Изменить + -

Сетевые интерфейсы (с привязкой к vNIC профилю)
nic1 Выберите пункт... + -

Скрыть расширенные настройки

OK Заккрыть

- Остальные параметры оставьте по умолчанию, создайте виртуальную машину. Когда процесс создания VM закончится, создайте шаблон на ее основе:
 - Выберите в списке созданную VM и перейдите в окно создания нового шаблона.



- Заполните параметры шаблона, обязательным является только название. Включите параметр *Запечатать шаблон*:

Новый шаблон ✕

Имя Deckhouse-Node

Описание

Комментарий

Кластер Default/Default

Профиль ЦП Default

☐ Создать, как подверсию шаблона

Выделение дискового пространства:

Имя	Виртуальный размер	Формат	Цель	Профиль диска
Astra-Linux-Special-Edi	2 GiB	QCOW2	Local (1638 GiB св	Local

☒ Разрешить всем пользователям доступ к шаблону

☐ Копировать разрешения VM

☒ Запечатать шаблон (только Linux)

OK Заккрыть

На время создания шаблона виртуальная машина и диск cloud-образа перейдут в состояние *Образ заблокирован*. Проверить что шаблон был успешно создан можно в разделе *Ресурсы -> Шаблоны*:

Имя	Версия	Комментарий	Дата создания	Состояние	Запечатано
Deckhouse-Node			14 февр. 2024 г., 16:23:...	ОК	Да

Custom Resources

i

Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

ZvirtInstanceClass

Scope: Cluster

- v1
- v1alpha1

Параметры группы Zvirt servers, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- spec**
объект

Обязательный параметр

- spec.memory**
целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам zVirt VirtualMachines.

Пример:

```
memory: 8192
```

- spec.numCPUs**
целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам zVirt.

Пример:

```
numCPUs: 2
```

- spec.rootDiskSizeGb**
целочисленный

Размер корневого диска в GiB для использования в виртуальных машинах zVirt.

По умолчанию: `30`

Пример:

```
rootDiskSizeGb: 30
```

- spec.template**
строка

Обязательный параметр

Имя шаблона, из которого будут клонированы VM.

Пример:

```
template: debian-bookworm
```

- spec.vnicProfileID**
строка

Обязательный параметр

vNIC профиль id.

Шаблон: `^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}$`

Пример:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

Параметры группы Zvirt servers, которые будет использовать `machine-controller-manager` (модуль `node-manager`).

На этот ресурс ссылается ресурс `CloudInstanceClass` модуля `node-manager`.

- **spec**
объект
Обязательный параметр

- **spec.memory**
целочисленный

Обязательный параметр

Память в MiB для выделения виртуальным машинам zVirt VirtualMachines.

Пример:

```
memory: 8192
```

- **spec.numCPUs**
целочисленный

Обязательный параметр

Количество vCPU, выделяемых виртуальным машинам zVirt.

Пример:

```
numCPUs: 2
```

- **spec.rootDiskSizeGb**
целочисленный

Размер корневого диска в GiB для использования в виртуальных машинах zVirt.

По умолчанию: `30`

Пример:

```
rootDiskSizeGb: 30
```

- **spec.template**
строка

Обязательный параметр

Имя шаблона, из которого будут клонированы VM.

Пример:

```
template: debian-bookworm
```

- **spec.vnicProfileID**
строка

Обязательный параметр

vNIC профиль id.

Шаблон: `^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}$`

Пример:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **status**
объект
 - **status.nodeGroupConsumers**
массив строк

FAQ

Как получить vNicProfileId

VNicProfileId можно получить путем запроса к zVirt API

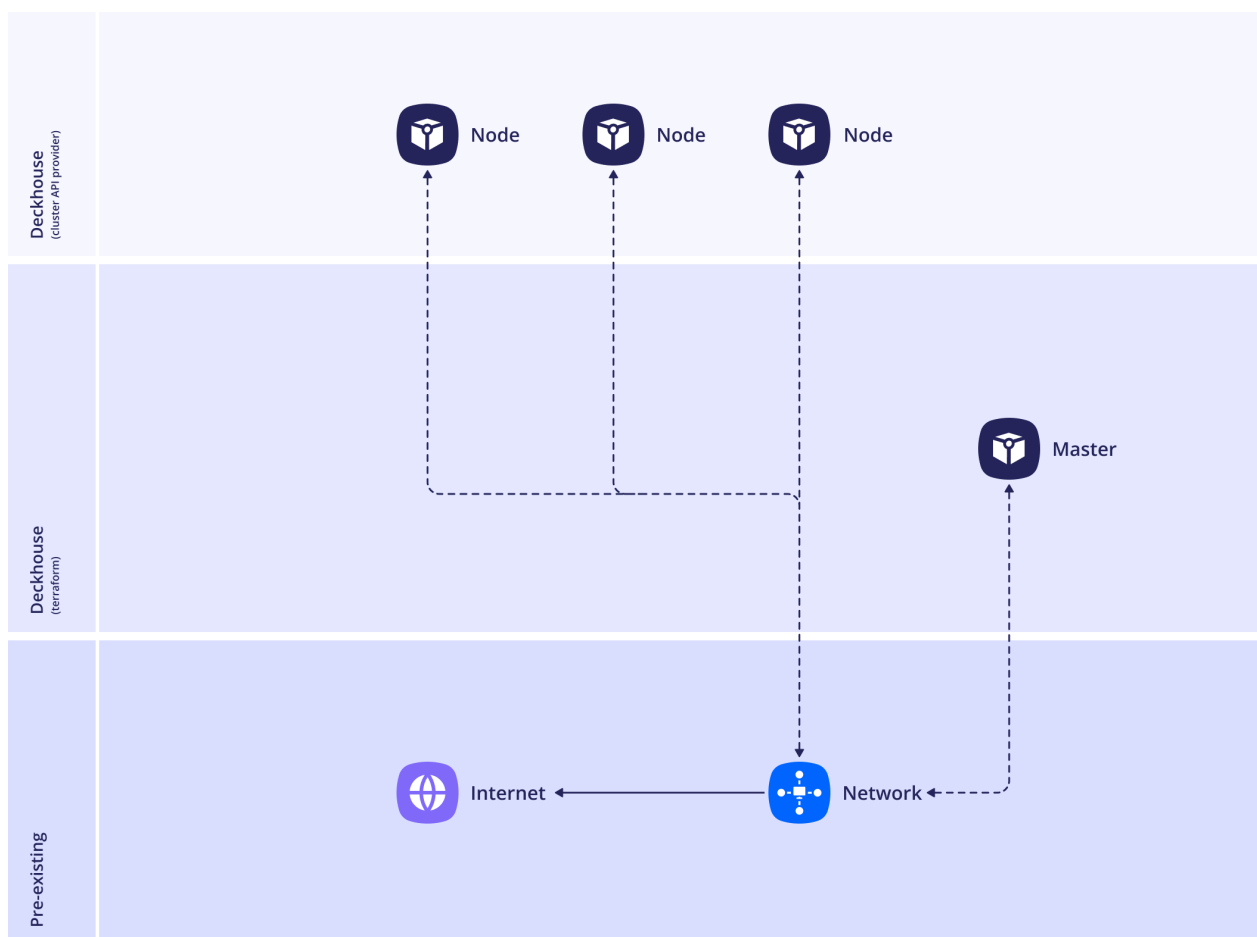
```
curl -u "<имя пользователя>@<профиль>:<пароль>" -X GET https://<zVirt API URL>/vnicprofiles
```

Пример ответа

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vnic_profiles>
  <vnic_profile href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-8594eafd5a86"
id="49bb4594-0cd4-4eb7-8288-8594eafd5a86">
    <name>vm-net-01</name>
    <link href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-
8594eafd5a86/permissions" rel="permissions"/>
    <pass_through>
      <mode>disabled</mode>
    </pass_through>
    <port_mirroring>false</port_mirroring>
    <network href="/ovirt-engine/api/networks/74a741c9-0d40-4008-8e58-1c903ee6eba7"
id="74a741c9-0d40-4008-8e58-1c903ee6eba7"/>
  </vnic_profile>
  ...
</vnic_profiles>
```

Cloud provider - zVirt: схемы размещения

Standard



Пример конфигурации схемы размещения:

```

---
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
sshPublicKey: "ssh-rsa AAAABBBB"
clusterID: "b46372e7-0d52-40c7-9bbf-fda31e187088"
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    rootDiskSizeGb: 40
    template: ALT-p10
    vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
    storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"

```

Модуль cni-cilium

Модуль `cni-cilium` обеспечивает работу сети в кластере. Основан на проекте Cilium.

Ограничения

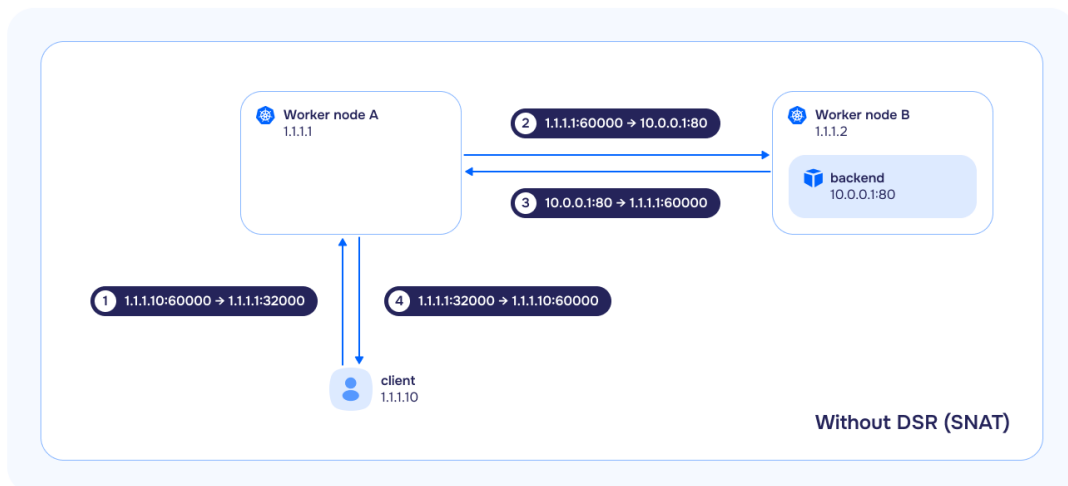
1. Сервисам с типом `NodePort` и `LoadBalancer` не подходят эндпоинты в LB-режиме `DSR`, работающие с `hostNetwork`. Если это необходимо, переключитесь на режим `SNAT`.
2. Поды `HostPort` связываются только с одним IP-адресом. Если в ОС есть несколько интерфейсов/IP, Cilium выберет один, предпочитая «серые» «белым».
3. Для обеспечения стабильной работы `cni-cilium` на узлах кластера отключите Elastic Agent или ограничьте доступ этого агента к серверу управления Elastic. В состав Elastic Agent входит компонент Elastic Endpoint, который использует технологию Extended Berkeley Packet Filter (eBPF) на узлах кластера и может удалять критически важные eBPF-программы, необходимые для корректной работы `cni-cilium`. Детальная информация и обсуждение проблемы доступны в публикациях проектов Cilium и Elastic.
4. Требования к ядру:
 - ядро Linux версии не ниже 5.8 для работы модуля `cni-cilium` и его совместной работы с модулями `istio`, `openvpn`, `node-local-dns`.
5. Совместимость с ОС:
 - Ubuntu:
 - несовместим с версией 18.04;
 - для работы с версией 20.04 необходима установка ядра HWE.
 - Astra Linux:
 - несовместим с изданием «Смоленск».
 - CentOS:
 - для версий 7 и 8 необходимо новое ядро из репозитория.

Обработка внешнего трафика в разных режимах работы `bpfLB` (замена `kube-proxy` от Cilium)

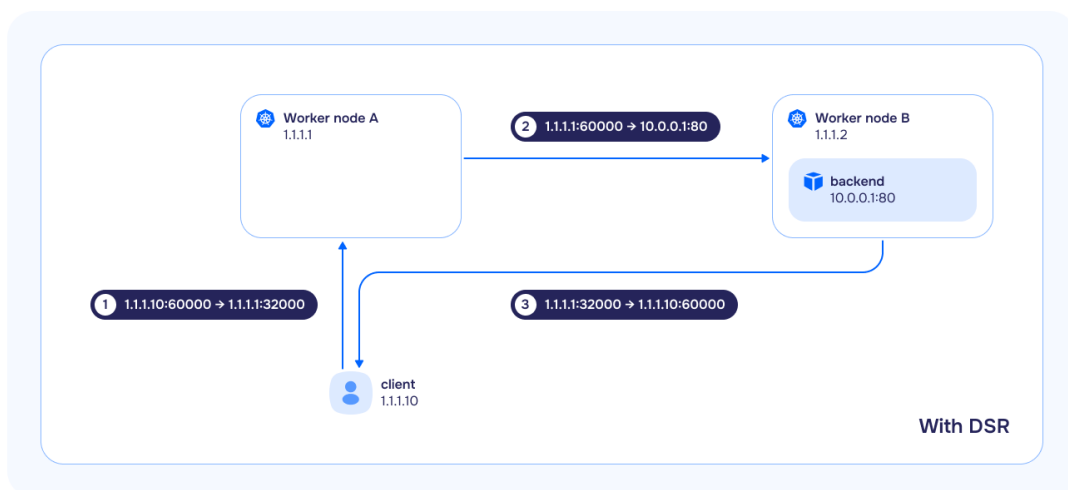
В Kubernetes обычно используются схемы, где трафик приходит на балансировщик, который распределяет его между многими серверами. Через балансировщик проходит и входящий, и исходящий трафик. Таким образом, общая пропускная способность ограничена ресурсами и шириной канала балансировщика. Для оптимизации трафика и разгрузки балансировщика и был придуман механизм `DSR`, в котором входящие пакеты проходят через балансировщик, а исходящие идут напрямую с терминирующих серверов. Так как обычно ответы имеют много больший размер чем запросы, то такой подход позволяет значительно увеличить общую пропускную способность схемы.

В модуле возможен выбор режима работы, влияющий на поведение `Service` с типом `NodePort` и `LoadBalancer`:

- **SNAT** (Source Network Address Translation) — один из подвидов NAT, при котором для каждого исходящего пакета происходит трансляция IP-адреса источника в IP-адрес шлюза из целевой подсети, а входящие пакеты, проходящие через шлюз, транслируются обратно на основе таблицы трансляций. В этом режиме `bpfLB` полностью повторяет логику работы `kube-proxy`:
 - если в `Service` указан `externalTrafficPolicy: Local`, то трафик будет передаваться и балансироваться только в те целевые поды, которые запущены на том же узле, на который этот трафик пришел. Если целевой под не запущен на этом узле, то трафик будет отброшен.
 - если в `Service` указан `externalTrafficPolicy: Cluster`, то трафик будет передаваться и балансироваться во все целевые поды в кластере. При этом, если целевые поды находятся на других узлах, то при передаче трафика на них будет произведен SNAT (IP-адрес источника будет заменен на InternalIP узла).



- **DSR** - (Direct Server Return) — метод, при котором весь входящий трафик проходит через балансировщик нагрузки, а весь исходящий трафик обходит его. Такой метод используется вместо SNAT. Часто ответы имеют много больший размер чем запросы и DSR позволяет значительно увеличить общую пропускную способность схемы:
 - если в Service указан `externalTrafficPolicy: Local`, то поведение абсолютно аналогично kube-proxy и bpflb в режиме SNAT.
 - если в Service указан `externalTrafficPolicy: Cluster`, то трафик так же будет передаваться и балансироваться во все целевые поды в кластере. При этом важно учитывать следующие особенности:
 - если целевые поды находятся на других узлах, то при передаче на них входящего трафика будет сохранен IP-адрес источника;
 - исходящий трафик пойдет прямо с узла, на котором был запущен целевой под;
 - IP-адрес источника будет заменен на внешний IP-адрес узла, на которую изначально пришел входящий запрос.



В случае использования режима DSR и Service с `externalTrafficPolicy: Cluster` требуются дополнительные настройки сетевого окружения. Сетевое оборудование должно быть готово к асимметричному прохождению трафика: отключены или настроены соответствующим образом средства фильтрации IP адресов на входе в сеть (uRPF, sourceGuard и т.п.).

- **Hybrid** — в данном режиме TCP-трафик обрабатывается в режиме DSR, а UDP — в режиме SNAT.

Использование CiliumClusterwideNetworkPolicies



Использование CiliumClusterwideNetworkPolicies при отсутствии опции `policyAuditMode` в настройках модуля cni-cilium может привести к некорректной работе Control plane или потере доступа ко всем узлам кластера по SSH.

Для использования CiliumClusterwideNetworkPolicies выполните следующие шаги:

1. Примените первичный набор объектов CiliumClusterwideNetworkPolicy. Для этого в настройки модуля cni-cilium добавьте конфигурационную опцию policyAuditMode со значением true. Опция policyAuditMode может быть удалена после применения всех CiliumClusterwideNetworkPolicy-объектов и проверки корректности их работы в Hubble UI.
2. Примените правило политики сетевой безопасности:

```
apiVersion: "cilium.io/v2"
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: "allow-control-plane-connectivity"
spec:
  ingress:
    - fromEntities:
      - kube-apiserver
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/control-plane: ""
```

В случае, если CiliumClusterwideNetworkPolicies не будут использованы, Control plane может некорректно работать до одной минуты во время перезагрузки cilium-agent-подов. Это происходит из-за сброса Conntrack-таблицы. Привязка к entity kube-apiserver позволяет избежать проблемы.

Смена режима работы Cilium

При смене режима работы Cilium (параметр tunnelMode) с Disabled на VXLAN или обратно, необходимо перезагрузить все узлы, иначе возможны проблемы с доступностью подов.

Выключение модуля kube-proxy

Cilium полностью заменяет собой функционал модуля kube-proxy, поэтому kube-proxy автоматически отключается при включении модуля cni-cilium.

Использование выборочного алгоритма балансировки нагрузки для сервисов

В Deckhouse Kubernetes Platform для балансировки нагрузки трафика сервисов можно применять следующие алгоритмы:

- **Random** — случайный выбор бэкенда для каждого соединения. Прост в реализации, но не всегда обеспечивает равномерное распределение.
- **Maglev** — использует консистентное хеширование для равномерного распределения трафика, подходит для масштабных сервисов с множеством бэкендов, которые часто ротируются.
- **Least Connections** — направляет трафик на бэкенд с наименьшим количеством активных соединений, оптимизируя нагрузку для приложений с длительными соединениями.

По умолчанию для всех сервисов задан алгоритм балансировки **Random**. Однако Deckhouse позволяет переопределять алгоритм для отдельных сервисов. Чтобы использовать выборочный алгоритм балансировки для конкретного сервиса, выполните следующие шаги:

- Отредактируйте конфигурацию модуля cni-cilium в Deckhouse, включив параметр extraLoadBalancerAlgorithmsEnabled. Это активирует поддержку аннотаций сервисов для выборочных алгоритмов.
- В манифесте сервиса укажите аннотацию service.cilium.io/lb-algorithm с одним из значений: random, maglev или least-conn.



Для корректной работы данного механизма требуется версия ядра Linux 5.15 и выше.

Использование Egress Gateway

Egress Gateway в Deckhouse Kubernetes Platform может быть использован в одном из двух режимов: Базовый и Режим с Virtual IP. Для выбора режима используйте ресурс EgressGateway (параметр spec.sourceIP.node).

Базовый режим

Используются предварительно настроенные IP-адреса на egress-узлах.

Режим с Virtual IP

Реализована возможность динамически назначать дополнительные IP-адреса узлам.

Экспорт данных из Hubble

Deckhouse Kubernetes Platform позволяет настраивать экспорт данных из Hubble, работающего внутри агентов Cilium, с помощью кластерного пользовательского ресурса HubbleMonitoringConfig: Чтобы включить экспорт,

создайте ресурс HubbleMonitoringConfig.



Создание или изменение ресурса HubbleMonitoringConfig приведёт к **перезапуску всех агентов Cilium** в кластере.

Настройки

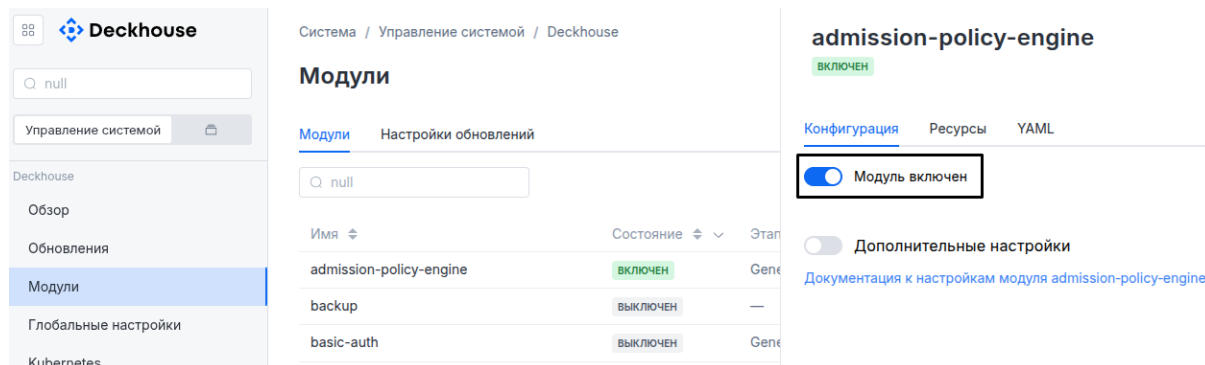
В модуле реализовано 8 алертов.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-cilium`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cni-cilium`:

```
d8 system module enable cni-cilium
```

- **С помощью ModuleConfig `cni-cilium`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cni-cilium` (создайте его, при необходимости).

Пример манифеста для включения модуля `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-cilium`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `cni-cilium` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-cilium`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cni-cilium` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cni-cilium
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cni-cilium`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.activeL2NeighborDiscoveryEnabled**
булевый

Включает механизм `L2 neighbor discovery`, который помогает гарантировать актуальность записей в таблице ARP на всех узлах кластера.

По умолчанию: `true`

- **settings.bpfLBMode**
строка

Режим работы балансировщика eBPF:

- **SNAT** — трафик от клиента до пода (и обратно) проходит через NAT, соответственно теряется адрес отправителя.
- **DSR** — трафик от клиента до пода проходит с сохранением адреса отправителя, а обратно - согласно правилам роутинга (минуя балансировщик). Этот режим экономит сетевой трафик, уменьшает задержки, но работает только для TCP трафика.
- **Hybrid** — TCP трафик обрабатывается в режиме DSR, а UDP - в режиме SNAT.

По умолчанию: `Hybrid`

Допустимые значения: `SNAT`, `Hybrid`, `DSR`

◦ **settings.createNodeRoutes**

булевый

Включает создание маршрутов к подам на других узлах.

Все узлы должны находиться в одной сети L2.

◦ **settings.debugLogging**

булевый

Включает отладочный уровень логирования для компонентов Cilium.

По умолчанию: `false`

◦ **settings.encryption**

объект

Настройки шифрования CNI Cilium.

▪ **settings.encryption.mode**

строка

Режим шифрования трафика:

- `Disabled` — шифрование трафика отключено.
- `WireGuard` — включено шифрование трафика между подами с использованием WireGuard.

i Внимание! Перед использованием шифрования с помощью WireGuard, пожалуйста убедитесь, что ядро Linux, используемое на узлах кластера, поддерживает WireGuard (указан параметр конфигурации ядра `CONFIG_WIREGUARD=m` для Linux 5.6 и старше, или установлен внешний модуль WireGuard для более ранних версий ядра).

i Внимание! Использование шифрования с помощью WireGuard может повлечь значительное потребление CPU подами агента Cilium.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `WireGuard`

▪ **settings.encryption.nodeToNodeModeEnabled**

булевый

Включение шифрования трафика между узлами и подами работающими в `HostNetwork`.

i Внимание! Межузловой трафик к узлам control-plane или от них остается незашифрованным, для предотвращения возможных проблем сетевого взаимодействия в процессе обмена ключами шифрования WireGuard. При этом, сетевой трафик между системными компонентами платформы остаётся зашифрованным на прикладном уровне.

i Внимание! Когда этот режим активен, из `PodNetwork` не доступны `NodePort` and `HostPort`.

По умолчанию: `false`

◦ **settings.exclusiveCNIPlugin**

булевый

Предотвращает включение дополнительных CNI-плагинов на узлах кластера. Если настройка модуля `Istio dataPlane.trafficRedirectionSetupMode` установлена в значение `CNIPlugin`, то данная опция автоматически будет установлена в значение `false`.

Явно отключать эту опцию рекомендуется только при использовании дополнительного CNI или специфичного CNI-плагина в кластере. Примером такого использования может быть CNI-плагин Linkerd.

По умолчанию: `true`

◦ **settings.extraLoadBalancerAlgorithmsEnabled**

булевый

Позволяет избирательно переопределить алгоритм балансировки с помощью eBPF между бэкэндами для отдельных сервисов. Для переопределения используется аннотация `service.cilium.io/lb-algorithm` с одним из значений: `random`, `maglev` или `least-conn`.

i Внимание! Данная опция для работы требует на узлах версию ядра Linux не ниже 5.15.

По умолчанию: `false`

◦ **settings.labelsRegex**
массив строк

Cilium создает идентификаторы безопасности основываясь на лейблах сущностей k8s, чем больше лейблов участвует в этом процессе - тем более детализованно можно настроить доступы. Но в кластерах больших объёмов излишняя детализация может создать большую нагрузку, и при помощи опции `labelsRegex` можно более явно указать - какие лейблы будут использоваться для политик безопасности, а какими следует пренебречь. Подробнее об уменьшении количества создаваемых identity можно прочитать в документации

Каждый лейбл должен задаваться RegExp-выражением в формате YAML quoted string с экранированием спецсимволов.

Пример:

```
labelsRegex:
- k8s:!app\.kubernetes\.io
- k8s:io\.cilium\.k8s\.policy
```

◦ **settings.loadBalancerICMPReplyEnabled**
булевый

Включает генерацию ICMP Echo Reply для IP-адресов сервисов типа LoadBalancer.

При включении узлы отвечают на ICMP Echo Request (ping), адресованные внешним IP-адресам LoadBalancer.

По умолчанию: `false`

◦ **settings.loadBalancerSourceRangeAllTypes**
булевый

Включает проверку source IP для всех типов сервисов, а не только для LoadBalancer. К соответствующему NodePort или ClusterIP также будет применен фильтр CIDR. Для получения более подробной информации, пожалуйста, перейдите по ссылке.

По умолчанию: `true`

◦ **settings.masqueradeMode**
строка

Режим работы SNAT для трафика подов, исходящего из кластера.

- **BPF** — использовать BPF. Базовый режим работы.

В данном режиме сетевые запросы из подов не будут подвержены процедуре Source NAT, если IP-адрес назначения принадлежит `podSubnetCIDR` или `InternalIP/ExternalIP` любого из узлов кластера.

- **Netfilter** — использовать Netfilter(iptables/nf_tables).

В данном режиме сетевые запросы из подов не будут подвержены процедуре Source NAT, если IP-адрес назначения принадлежит `podSubnetCIDR` или `InternalIP` любого из узлов кластера.

i В режиме BPF, если на узле кластера указан ExternalIP, отличный от InternalIP, но этот IP-адрес не назначен на ни на один интерфейс, то он будет не доступен с соседних узлов из сети подов. Режим Netfilter позволяет обойти это ограничение. Однако при этом некоторые другие функции перестают работать, в частности EgressGateway.

По умолчанию: `BPF`

Допустимые значения: `Netfilter`, `BPF`

◦ **settings.policyAuditMode**
булевый

Отключает влияние сетевых политик на трафик. Сетевые запросы, попавшие под запрет политик, просто логируются.

По умолчанию: `false`

◦ **settings.resourcesManagement**

Настройки запросов (requests) и ограничений (limits) использования CPU и памяти подами агента cilium.

Примеры:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

■ **settings.resourcesManagement.mode**

строка

Режим управления ресурсами:

- **Static** — классический, с помощью указания запросов/ограничений (requests/limits). Настраивается в секции параметров static;
- **VPA** — с помощью VPA. Настраивается в секции параметров vpa.

По умолчанию: **VPA**

Допустимые значения: **VPA**, **Static**

■ **settings.resourcesManagement.static**

объект

Настройка управления ресурсами в режиме **Static**.

■ **settings.resourcesManagement.static.limits**

объект

Настройка ограничений (limits) использования CPU и памяти.

■ **settings.resourcesManagement.static.limits.cpu**

строка or число

Настройка ограничений использования CPU (CPU limits).

Шаблон: `^[0-9]+m?$`

■ **settings.resourcesManagement.static.limits.memory**

строка or число

Настройка ограничений использования памяти (memory limits).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **settings.resourcesManagement.static.requests**

объект

Настройки запросов ресурсов (requests) подов.

■ **settings.resourcesManagement.static.requests.cpu**

строка or число

Настройка запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.static.requests.memory**

строка or число

Настройка запроса памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa**
объект

Настройка управления ресурсами в режиме `VPA`.

- **settings.resourcesManagement.vpa.cpu**
объект

Настройки VPA при работе с CPU.

- **settings.resourcesManagement.vpa.cpu.limitRatio**
число

Коэффициент расчета ограничений использования CPU относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования CPU для пода (CPU limits).

Если параметр указан, то VPA, при пересчете запросов CPU (CPU requests) будет пересчитывать ограничения использования CPU (CPU limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.resourcesManagement.vpa.cpu.max**
строка or число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `4`

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.cpu.min**
строка or число

Минимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `100m`

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.memory**
объект

Настройка ограничений VPA (limits) при работе с памятью.

- **settings.resourcesManagement.vpa.memory.limitRatio**
число

Коэффициент расчета ограничений использования памяти относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования памяти для пода (memory limits).

Если параметр указан, то VPA, при пересчете запросов памяти (memory requests) будет пересчитывать ограничения использования памяти (memory limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.resourcesManagement.vpa.memory.max**
строка or число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: `4Gi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.memory.min**
строка or число

Минимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: `128Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **settings.resourcesManagement.vpa.mode**

строка

Режим работы VPA.

- **Initial** — VPA устанавливает начальные значения requests/limits при создании пода. В дальнейшем значения ресурсов не изменяются автоматически.
- **InPlaceOrRecreate** — VPA пытается изменить ресурсы пода «на месте» (in-place), если это поддерживается кластером. Если in-place обновление невозможно, под будет пересоздан.
- **Auto** — VPA автоматически пересоздаёт поды для применения новых значений ресурсов. Начиная с версии Deckhouse 1.75 данный режим считается устаревшим. Рекомендуется использовать **InPlaceOrRecreate**.

По умолчанию: `Initial`

Допустимые значения: `Initial`, `InPlaceOrRecreate`, `Auto`

○ **settings.sctpEnabled**

булевый

Включение поддержки протокола SCTP в Cilium (как для входящего, так и исходящего трафика).

i **Внимание.** Cilium реализует только базовую поддержку протокола SCTP. При его использовании, следует учесть некоторые ограничения:

- Не реализованы следующие функции: `SCTP multi-homing`, `NetworkPolicies` для `pod-to-VIP`.
- Для входящего SCTP-трафика также не поддерживается перезапись портов в пакетах. Это означает, что приложения могут быть опубликованы либо через `HostPort`, либо через `Service`, в описании которого `targetPort` ДОЛЖЕН совпадать с `port`. В противном случае пакеты будут отброшены.
- Для исходящего SCTP-трафика по умолчанию поддерживается только 1:1 SNAT. В этом режиме подменяется только `src_ip`, а `src_port` остается оригинальным. Если же такой `src_port` уже занят на узле, то исходящий пакет будет отброшен. Чтобы также подменялся `src_port`, необходимо переключить параметр `masqueradeMode` в режим `Netfilter`.

По умолчанию: `false`

○ **settings.svcSourceRangeCheck**

булевый

Для сервисов с типом `loadBalancer` включает проверку source IP на его соответствие `loadBalancer.sourceRanges`. Проверка не производится, если доступ производится изнутри кластера или параметр `sourceRanges` пуст/не указан. Некоторые облачные провайдеры в своих реализациях `loadBalancer` поддерживают фильтрацию входящего трафика на основе параметра `sourceRanges`, в таком случае документация по cilium рекомендует отключать проверку `svcSourceRangeCheck`.

Не включайте, если перед сервисами есть другой балансировщик или какие-либо виды SNAT.

По умолчанию: `false`

○ **settings.tunnelMode**

строка

Режим работы туннеля.

i **Внимание!** После изменения параметра необходимо перезагрузить все узлы, иначе могут быть проблемы с доступностью подов.

i **Внимание!** Режим VXLAN не совместим со следующими режимами `bpfLBMode`: `Hybrid`, `DSR`. Если `tunnelMode` настроен на `VXLAN`, то `bpfLBMode` будет переопределён на `SNAT` автоматически.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `VXLAN`

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

EgressGatewayPolicy

Scope: Cluster
Version: v1alpha1

Интерфейс для настройки политик перенаправления прикладного трафика на определённые egress-шлюзы, описанные через интерфейс EgressGateway.

- **spec**
объект

Обязательный параметр

- **spec.destinationCIDRs**
массив строк

Доступно в редакциях: SE+, EE

Сетевые запросы в сторону данных подсетей будут маршрутизированы через egress-шлюз.

- **spec.egressGatewayName**
строка

Обязательный параметр

Доступно в редакциях: SE+, EE

Имя ресурса EgressGateway, в котором описан egress-шлюз для обработки трафика.

- **spec.excludedCIDRs**
массив строк

Доступно в редакциях: SE+, EE

Сетевые диапазоны, которые требуется исключить из диапазонов, описанных в `destinationCIDRs`. Обратите внимание, что исключать служебные диапазоны кластера (`podSubnetCIDR` и `serviceSubnetCIDR`) не требуется.

- **spec.selectors**
массив объектов

Доступно в редакциях: SE+, EE

Селекторы прикладных подов. Их трафик будет маршрутизирован через соответствующий egress-шлюз. Для обозначения namespace используйте псевдо-лейбл `io.kubernetes.pod.namespace`.

- **spec.selectors.podSelector**
объект

Обязательный параметр

Доступно в редакциях: SE+, EE

Настройка фильтра лейблов по ресурсам.

Если указаны одновременно `matchExpressions` и `matchLabels`, учитываются оба фильтра (операция И).

Пустое значение `labelSelector` соответствует всем объектам. Нулевое — никаким.

- **spec.selectors.podSelector.matchExpressions**
массив объектов

Доступно в редакциях: SE+, EE

Список фильтров на основе выражений.

Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Фильтр на основе выражения.

- **spec.selectors.podSelector.matchExpressions.key**
строка

Обязательный параметр

Доступно в редакциях: SE+, EE

Имя лейбла.

- **spec.selectors.podSelector.matchExpressions.operator**
строка

Обязательный параметр

Доступно в редакциях: SE+, EE

Оператор сравнения.

- **spec.selectors.podSelector.matchExpressions.values**
массив строк

Доступно в редакциях: SE+, EE

Значение лейбла.

- **spec.selectors.podSelector.matchLabels**
объект

Доступно в редакциях: SE+, EE

Фильтр на основе совпадения/несовпадения лейблов.

- **status**
объект

EgressGateway

Scope: Cluster
Version: v1alpha1

Интерфейс для настройки отказоустойчивого egress-шлюза на основе группы узлов.

Настроенные egress-шлюзы можно использовать в политиках исходящего трафика EgressGatewayPolicy.

- **spec**
объект
 - **spec.nodeSelector**
объект

Доступно в редакциях: SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Селектор для группы узлов, которые будут осуществлять передачу сетевых запросов на внешние сервисы.

Среди данных узлов будут выявлены пригодные к работе и один из них будет назначен активным. Признаки пригодного узла:

- Узел в состоянии Ready.
- Узел не находится в состоянии технического обслуживания (cordon).
- cilium-agent на узле в состоянии Ready.

Разные EgressGateway могут использовать для работы общие узлы, при этом активные узлы будут выбираться независимо, тем самым распределяя нагрузку между ними.

- **spec.sourceIP**
объект
 - **spec.sourceIP.mode**
строка

Обязательный параметр

Доступно в редакциях: SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Метод определения исходящего IP-адреса, который будет назначен на исходящие запросы через данный шлюз.

Возможны следующие варианты:

- **PrimaryIPFromEgressGatewayNodeInterface** (базовый режим) — в качестве IP-адреса будет использоваться primary IP-адрес на *публичном* сетевом интерфейсе узла.

Особенности:

- При выходе из строя активного узла и назначении нового, IP-адрес отправителя в сетевых пакетах поменяется.
- Предварительно необходимо настроить сетевую подсистему на всех egress-узлах:
 - у *публичных* интерфейсов должно быть одинаковое имя (например, eth1),
 - должны быть настроены все необходимые маршруты для доступа на все внешние публичные сервисы.

- **VirtualIPAddress** (режим с Virtual IP) — явное указание исходящего IP-адреса.

Особенности:

- При выходе из строя активного узла, IP-адрес отправителя в сетевых пакетах не поменяется.
- Предварительно необходимо настроить сетевую подсистему на всех egress-узлах:
 - должны быть настроены все необходимые маршруты для доступа на все внешние публичные сервисы,
 - *публичный* интерфейс должен быть подготовлен к автоматической настройке *виртуального* IP в качестве secondary IP-адреса (при назначении egress-узла в качестве активного, адрес не будет отображаться в списке IP на *публичном* интерфейсе, но при этом, узел будет эмулировать его наличие с помощью ARP-ответов).

Допустимые значения: **VirtualIPAddress**, **PrimaryIPFromEgressGatewayNodeInterface**

- **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface**
объект

Используется для базового режима (если `mode: PrimaryIPFromEgressGatewayNodeInterface`).

- **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface.interfaceName**
строка

Доступно в редакциях: **SE+, EE, CSE Lite (1.73), CSE Pro (1.73)**

Имя интерфейса на egress-узлах, через который осуществляется доступ к внешним сервисам.

- **spec.sourceIP.virtualIPAddress**
объект

Используется для режима Virtual IP (если `mode: VirtualIPAddress`).

- **spec.sourceIP.virtualIPAddress.interfaces**
массив строк

Доступно в редакциях: **SE+, EE, CSE Lite (1.73), CSE Pro (1.73)**

Список сетевых интерфейсов, на которых будет осуществляться имитация *виртуального* IP-адреса.

По умолчанию: `[]`

- **spec.sourceIP.virtualIPAddress.ip**
строка

Доступно в редакциях: **SE+, EE, CSE Lite (1.73), CSE Pro (1.73)**

Виртуальный IP-адрес, который будет назначен на исходящие запросы через egress-шлюз.

- **status**

объект

- **status.activeNodeName**
строка

Имя активного в данный момент узла.

- **status.conditions**

массив объектов

- **status.conditions.lastHeartbeatTime**
строка
- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**
строка

Обязательный параметр

message is a human readable message indicating details about the transition. This may be an empty string.

Максимальная длина: `32768`

- **status.conditions.observedGeneration**
целочисленный

observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

Допустимые значения: `0 <= X`

- **status.conditions.reason**
строка

Обязательный параметр

reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

Шаблон: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Длина: `1..1024`

- **status.conditions.status**
строка

Обязательный параметр

status of the condition, one of True, False, Unknown.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**
строка

Обязательный параметр

Type of condition in CamelCase or in foo.example.com/CamelCase.

Many .condition.type values are consistent across resources like Available, but because arbitrary conditions can be useful (see .node.status.conditions), the ability to deconflict is important. The regex it matches is (dns1123SubdomainFmt/)?(qualifiedNameFmt)

Шаблон: `^([a-z0-9]([a-z0-9]*[a-z0-9])?(\.[a-z0-9]([a-z0-9]*[a-z0-9])?)*\/)?([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

Максимальная длина: `316`

- **status.observedGeneration**
целочисленный
- **status.readyNodes**
целочисленный

Количество узлов, готовых осуществлять функции egress-шлюза.

HubbleMonitoringConfig

Scope: Cluster

Version: v1alpha1

- **spec**
объект

Обязательный параметр

- **spec.extendedMetrics**
объект

Конфигурация для экспорта расширенных метрик с агентов Cilium.

Расширенные метрики позволяют получить более глубокое понимание поведения сети кластера.

Внимание. Изменение этого параметра приведет к перезапуску агентов Cilium.

Для получения дополнительной информации смотрите документацию Cilium.

Пример:

```
extendedMetrics:
  enabled: true
  collectors:
    - name: drop
      contextOptions:
        labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,destination_pod
    - name: flow
```

- **spec.extendedMetrics.collectors**
массив объектов

Список расширенных метрик Hubble для экспорта.

Каждый коллектор представляет собой определённый тип метрики.

Для получения дополнительной информации смотрите документацию Cilium.

Пример:

```
collectors:
  - name: drop
    contextOptions:
      labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,destination_pod
  - name: flow
```

- **spec.extendedMetrics.collectors.contextOptions**
строка

Необязательные параметры контекста для метрик Hubble.

Определяет, по каким параметрам будут детализироваться метрики, которые формирует коллектор. Если не указать `contextOptions`, коллектор вернёт набор метрик по умолчанию, агрегированный в целом, без разбивки по отдельным параметрам. Например, для коллектора `drop` пустой `contextOptions` вернёт общую статистику дропов без различия конкретного источника и назначения. Если задать `contextOptions`, значения метрик будут группироваться по выбранным измерениям, например по Pod-ам источника и назначения, namespace и лейблам.

i **Внимание.** Большое количество лейблов в метриках может привести к снижению производительности Prometheus.

Для получения дополнительной информации о поддерживаемых параметрах смотрите документацию Cilium.

- **spec.extendedMetrics.collectors.name**
строка

Название расширенной метрики для экспорта.

Допустимые значения: `dns`, `drop`, `flow`, `flows-to-world`, `http`, `httpV2`, `icmp`, `kafka`, `tcp`

- **spec.extendedMetrics.enabled**
булевый

Включает экспорт расширенных метрик с агентов Cilium.

По умолчанию: `false`

- **spec.flowLogs**
объект

Конфигурация для экспорта журналов событий с агентов Cilium.

Журналы событий предоставляют детальную информацию о сетевых событиях Cilium.

i **Внимание.** Изменение этого параметра приведет к перезапуску агентов Cilium.

Для получения дополнительной информации смотрите документацию Cilium.

Пример:

```
flowLogs:
  enabled: true
  allowFilterList:
    - verdict:
        - DROPPED
        - ERROR
  denyFilterList:
    - source_pod:
        - kube-system/
    - destination_pod:
        - kube-system/
  fieldMaskList:
    - time
    - source.namespace
    - verdict
  fileSizeMB: 50
```

- **spec.flowLogs.allowFilterList**
массив объектов

Определяет фильтры, которые управляют тем, какие события экспортируются в лог. Экспортируются только события, которые соответствуют хотя бы одному из указанных объектов фильтра. Если фильтры не заданы, разрешены все события. Логическая семантика: - Внутри одного объекта фильтра все указанные поля объединяются по AND. - Между объектами фильтра в массиве фильтры объединяются по OR. Для получения дополнительной информации смотрите документацию Cilium.

- **spec.flowLogs.allowFilterList.destination_cluster_name**
массив строк

Имена кластеров назначения.

- **spec.flowLogs.allowFilterList.destination_fqdn**
массив строк

Полные доменные имена (FQDN) назначения.

- **Элемент массива**
строка

Максимальная длина: 255

- **spec.flowLogs.allowFilterList.destination_identity**
массив целых чисел

Идентификатор безопасности (security identity) назначения.

- **Элемент массива**
целочисленный

Допустимые значения: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.destination_ip**
массив строк

IP-адрес или CIDR назначения.

- **spec.flowLogs.allowFilterList.destination_label**
массив строк

Селекторы лейблов назначения.

Для получения дополнительной информации смотрите документацию Cilium.

Примеры:

```
destination_label: k8s:role=frontend
```

```
destination_label: reserved:kube-apiserver
```

- **spec.flowLogs.allowFilterList.destination_pod**
массив строк

Имена подов назначения в формате NamespacedName (namespace/pod).

Примеры:

```
destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
destination_pod: kube-system/
```

```
destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.allowFilterList.destination_port**
массив строк

Порт L4 назначения.

- **Элемент массива**
строка

Шаблон: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.destination_service**
массив строк

Имена сервисов назначения в формате NamespacedName (namespace/service).

Примеры:

```
destination_service: kube-system/kube-dns
```

```
destination_service: kube-system/
```

```
destination_service: "/kube-dns"
```

- **spec.flowLogs.allowFilterList.destination_workload**
массив объектов

Нагрузки назначения.

- **spec.flowLogs.allowFilterList.destination_workload.kind**
строка

Тип нагрузки.

- **spec.flowLogs.allowFilterList.destination_workload.name**
строка

Имя нагрузки.

- **spec.flowLogs.allowFilterList.dns_query**
массив строк

Шаблоны регулярных выражений для DNS-запросов.

- **spec.flowLogs.allowFilterList.drop_reason_desc**
массив строк

Описание причины отброса трафика (только для вердикта DROPPED).

- **Элемент массива**
строка

Допустимые значения: DROP_REASON_UNKNOWN, INVALID_SOURCE_MAC, INVALID_DESTINATION_MAC, INVALID_SOURCE_IP, POLICY_DENIED, INVALID_PACKET_DROPPED, CT_TRUNCATED_OR_INVALID_HEADER, CT_MISSING_TCP_ACK_FLAG, CT_UNKNOWN_L4_PROTOCOL, CT_CANNOT_CREATE_ENTRY_FROM_PACKET, UNSUPPORTED_L3_PROTOCOL, MISSED_TAIL_CALL, ERROR_WRITING_TO_PACKET, UNKNOWN_L4_PROTOCOL, UNKNOWN_ICMPV4_CODE, UNKNOWN_ICMPV4_TYPE, UNKNOWN_ICMPV6_CODE, UNKNOWN_ICMPV6_TYPE, ERROR_RETRIEVING_TUNNEL_KEY, ERROR_RETRIEVING_TUNNEL_OPTIONS, INVALID_GENEVE_OPTION, UNKNOWN_L3_TARGET_ADDRESS, STALE_OR_UNROUTABLE_IP, NO_MATCHING_LOCAL_CONTAINER_FOUND, ERROR_WHILE_CORRECTING_L3_CHECKSUM, ERROR_WHILE_CORRECTING_L4_CHECKSUM, CT_MAP_INSERTION_FAILED, INVALID_IPV6_EXTENSION_HEADER, IP_FRAGMENTATION_NOT_SUPPORTED, SERVICE_BACKEND_NOT_FOUND, NO_TUNNEL_OR_ENCAPSULATION_ENDPOINT, FAILED_TO_INSERT_INTO_PROXYMAP, REACHED_EDT_RATE_LIMITING_DROP_HORIZON, UNKNOWN_CONNECTION_TRACKING_STATE, LOCAL_HOST_IS_UNREACHABLE, NO_CONFIGURATION_AVAILABLE_TO_PERFORM_POLICY_DECISION, UNSUPPORTED_L2_PROTOCOL, NO_MAPPING_FOR_NAT_MASQUERADE, UNSUPPORTED_PROTOCOL_FOR_NAT_MASQUERADE, FIB_LOOKUP_FAILED, ENCAPSULATION_TRAFFIC_IS_PROHIBITED, INVALID_IDENTITY, UNKNOWN_SENDER, NAT_NOT_NEEDED, IS_A_CLUSTERIP, FIRST_LOGICAL_DATAGRAM_FRAGMENT_NOT_FOUND, FORBIDDEN_ICMPV6_MESSAGE, DENIED_BY_LB_SRC_RANGE_CHECK, SOCKET_LOOKUP_FAILED, SOCKET_ASSIGN_FAILED, PROXY_REDIRECTION_NOT_SUPPORTED_FOR_PROTOCOL, POLICY_DENY, VLAN_FILTERED, INVALID_VNI, INVALID_TC_BUFFER, NO_SID, MISSING_SRV6_STATE, NAT46, NAT64, AUTH_REQUIRED, CT_NO_MAP_FOUND, SNAT_NO_MAP_FOUND, INVALID_CLUSTER_ID, UNSUPPORTED_PROTOCOL_FOR_DSR_ENCAP, NO_EGRESS_GATEWAY, UNENCRYPTED_TRAFFIC, TTL_EXCEEDED, NO_NODE_ID, DROP_RATE_LIMITED, IGMP_HANDLED, IGMP_SUBSCRIBED, MULTICAST_HANDLED, DROP_HOST_NOT_READY, DROP_EP_NOT_READY, DROP_NO_EGRESS_IP

- **spec.flowLogs.allowFilterList.event_type**
массив объектов

Типы событий для фильтрации.

- **spec.flowLogs.allowFilterList.event_type.match_sub_type**
булевый

Нужно ли сопоставлять также sub_type.

- **spec.flowLogs.allowFilterList.event_type.sub_type**
целочисленный

Вторичный тип события.

- **spec.flowLogs.allowFilterList.event_type.type**
целочисленный

Основной тип события.

- **spec.flowLogs.allowFilterList.http_header**
массив объектов

HTTP-заголовки для сопоставления.

- **spec.flowLogs.allowFilterList.http_header.key**
строка

Ключ заголовка.

- **spec.flowLogs.allowFilterList.http_header.value**
строка

Значение заголовка.

- **spec.flowLogs.allowFilterList.http_method**
массив строк

HTTP-методы.

- **Элемент массива**
строка
 - Допустимые значения: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`
- **spec.flowLogs.allowFilterList.http_path**
массив строк
- Список регулярных выражений для сопоставления с HTTP-путем.
- **spec.flowLogs.allowFilterList.http_status_code**
массив строк
- Префиксы HTTP-кодов ответа.
- **Элемент массива**
строка
 - Шаблон: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`
- **spec.flowLogs.allowFilterList.http_url**
массив строк
- Список регулярных выражений для сопоставления с HTTP-адресом.
- **spec.flowLogs.allowFilterList.interface**
массив объектов
- Сетевой интерфейс, на котором был замечен поток.
- **spec.flowLogs.allowFilterList.interface.index**
целочисленный
 - Индекс интерфейса.
 - Допустимые значения: `0 <= X`
 - **spec.flowLogs.allowFilterList.interface.name**
строка
 - Имя интерфейса.
- **spec.flowLogs.allowFilterList.ip_version**
массив строк
- Версия IP.
- **Элемент массива**
строка
 - Допустимые значения: `IP_NOT_USED`, `IPv4`, `IPv6`
- **spec.flowLogs.allowFilterList.node_labels**
массив строк
- Селекторы лейблов узлов.
- Примеры:
- ```
node_labels: node-role.kubernetes.io/control-plane=
```
- ```
node_labels: topology.kubernetes.io/ru-central1-a
```
- **spec.flowLogs.allowFilterList.node_name**
массив строк
- Шаблоны имён узлов.
- **spec.flowLogs.allowFilterList.protocol**
массив строк
- Протокол L4 или L7.
- **spec.flowLogs.allowFilterList.source_cluster_name**
массив строк
- Имена кластеров источника.
- **spec.flowLogs.allowFilterList.source_fqdn**
массив строк
- Полные доменные имена (FQDN) источника.
- **Элемент массива**
строка
 - Максимальная длина: `255`

- **spec.flowLogs.allowFilterList.source_identity**

массив целых чисел

Идентификатор безопасности (security identity) источника.

- **Элемент массива**

целочисленный

Допустимые значения: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.source_ip**

массив строк

IP-адрес или CIDR источника.

- **spec.flowLogs.allowFilterList.source_ip_xlated**

массив строк

IP-адрес или CIDR источника после трансляции (SNAT).

- **spec.flowLogs.allowFilterList.source_label**

массив строк

Селекторы лейблов источника.

Для получения дополнительной информации смотрите документацию Cilium.

Примеры:

```
source_label: k8s:role=frontend
```

```
source_label: reserved:kube-apiserver
```

- **spec.flowLogs.allowFilterList.source_pod**

массив строк

Префиксы имён подов источника в формате NamespacedName (`namespace/pod`).

Примеры:

```
source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
source_pod: kube-system/
```

```
source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.allowFilterList.source_port**

массив строк

Порт L4 источника.

- **Элемент массива**

строка

Шаблон: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.source_service**

массив строк

Имена сервисов источника в формате NamespacedName (`namespace/service`).

Примеры:

```
source_service: kube-system/kube-dns
```

```
source_service: kube-system/
```

```
source_service: "/kube-dns"
```

- **spec.flowLogs.allowFilterList.source_workload**

массив объектов

Нагрузки источника.

- **spec.flowLogs.allowFilterList.source_workload.kind**

строка

Тип нагрузки.

- **spec.flowLogs.allowFilterList.source_workload.name**

строка

Имя нагрузки.

- **spec.flowLogs.allowFilterList.tcp_flags**

массив объектов

TCP-флаги для сопоставления.

- **spec.flowLogs.allowFilterList.tcp_flags.ACK**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.CWR**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.ECE**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.FIN**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.NS**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.PSH**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.RST**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.SYN**
булевый
- **spec.flowLogs.allowFilterList.tcp_flags.URG**
булевый

- **spec.flowLogs.allowFilterList.trace_id**

массив строк

ID трассировки.

- **spec.flowLogs.allowFilterList.traffic_direction**

массив строк

Направление трафика.

- **Элемент массива**
строка

Допустимые значения: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`

- **spec.flowLogs.allowFilterList.uuid**

массив строк

Фильтрация по UUID потоков (flow UUID).

- **spec.flowLogs.allowFilterList.verdict**

массив строк

Сопоставление по вердикту.

- **Элемент массива**
строка

Допустимые значения: `VERDICT_UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`

- **spec.flowLogs.denyFilterList**

массив объектов

Определяет фильтры, которые управляют тем, какие события явно исключаются из логирования. Любое событие, которое соответствует хотя бы одному из указанных объектов фильтра, будет запрещено к экспорту. Если фильтры не заданы, ничего не исключается. Логическая семантика: - Внутри одного объекта фильтра все указанные поля объединяются по AND. - Между объектами фильтра в массиве фильтры объединяются по OR. Для получения дополнительной информации смотрите документацию Cilium.

- **spec.flowLogs.denyFilterList.destination_cluster_name**

массив строк

Имена кластеров назначения.

- **spec.flowLogs.denyFilterList.destination_fqdn**

массив строк

Полные доменные имена (FQDN) назначения.

- **Элемент массива**
строка

Максимальная длина: 255

- **spec.flowLogs.denyFilterList.destination_identity**

массив целых чисел

Идентификатор безопасности (security identity) назначения.

- **Элемент массива**
целочисленный

Допустимые значения: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.destination_ip**
массив строк

IP-адрес или CIDR назначения.

- **spec.flowLogs.denyFilterList.destination_label**
массив строк

Селекторы лейблов назначения.

Для получения дополнительной информации смотрите документацию Cilium.

Примеры:

```
destination_label: k8s:role=frontend
```

```
destination_label: reserved:kube-apiserver
```

- **spec.flowLogs.denyFilterList.destination_pod**
массив строк

Имена подов назначения в формате NamespacedName (`namespace/pod`).

Примеры:

```
destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
destination_pod: kube-system/
```

```
destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.denyFilterList.destination_port**
массив строк

Порт L4 назначения.

- **Элемент массива**
строка

Шаблон: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.destination_service**
массив строк

Имена сервисов назначения в формате NamespacedName (`namespace/service`).

Примеры:

```
destination_service: kube-system/kube-dns
```

```
destination_service: kube-system/
```

```
destination_service: "/kube-dns"
```

- **spec.flowLogs.denyFilterList.destination_workload**
массив объектов

Нагрузки назначения.

- **spec.flowLogs.denyFilterList.destination_workload.kind**
строка

Тип нагрузки.

- **spec.flowLogs.denyFilterList.destination_workload.name**
строка

Имя нагрузки.

- **spec.flowLogs.denyFilterList.dns_query**
массив строк

Шаблоны регулярных выражений для DNS-запросов.

- **spec.flowLogs.denyFilterList.drop_reason_desc**

массив строк

Описание причины отброса трафика (только для вердикта DROPPED).

- **Элемент массива**
строка

Допустимые значения: DROP_REASON_UNKNOWN, INVALID_SOURCE_MAC, INVALID_DESTINATION_MAC, INVALID_SOURCE_IP, POLICY_DENIED, INVALID_PACKET_DROPPED, CT_TRUNCATED_OR_INVALID_HEADER, CT_MISSING_TCP_ACK_FLAG, CT_UNKNOWN_L4_PROTOCOL, CT_CANNOT_CREATE_ENTRY_FROM_PACKET, UNSUPPORTED_L3_PROTOCOL, MISSED_TAIL_CALL, ERROR_WRITING_TO_PACKET, UNKNOWN_L4_PROTOCOL, UNKNOWN_ICMPV4_CODE, UNKNOWN_ICMPV4_TYPE, UNKNOWN_ICMPV6_CODE, UNKNOWN_ICMPV6_TYPE, ERROR_RETRIEVING_TUNNEL_KEY, ERROR_RETRIEVING_TUNNEL_OPTIONS, INVALID_GENEVE_OPTION, UNKNOWN_L3_TARGET_ADDRESS, STALE_OR_UNROUTABLE_IP, NO_MATCHING_LOCAL_CONTAINER_FOUND, ERROR_WHILE_CORRECTING_L3_CHECKSUM, ERROR_WHILE_CORRECTING_L4_CHECKSUM, CT_MAP_INSERTION_FAILED, INVALID_IPV6_EXTENSION_HEADER, IP_FRAGMENTATION_NOT_SUPPORTED, SERVICE_BACKEND_NOT_FOUND, NO_TUNNEL_OR_ENCAPSULATION_ENDPOINT, FAILED_TO_INSERT_INTO_PROXYMAP, REACHED_EDT_RATE_LIMITING_DROP_HORIZON, UNKNOWN_CONNECTION_TRACKING_STATE, LOCAL_HOST_IS_UNREACHABLE, NO_CONFIGURATION_AVAILABLE_TO_PERFORM_POLICY_DECISION, UNSUPPORTED_L2_PROTOCOL, NO_MAPPING_FOR_NAT_MASQUERADE, UNSUPPORTED_PROTOCOL_FOR_NAT_MASQUERADE, FIB_LOOKUP_FAILED, ENCAPSULATION_TRAFFIC_IS_PROHIBITED, INVALID_IDENTITY, UNKNOWN_SENDER, NAT_NOT_NEEDED, IS_A_CLUSTERIP, FIRST_LOGICAL_DATAGRAM_FRAGMENT_NOT_FOUND, FORBIDDEN_ICMPV6_MESSAGE, DENIED_BY_LB_SRC_RANGE_CHECK, SOCKET_LOOKUP_FAILED, SOCKET_ASSIGN_FAILED, PROXY_REDIRECTION_NOT_SUPPORTED_FOR_PROTOCOL, POLICY_DENY, VLAN_FILTERED, INVALID_VNI, INVALID_TC_BUFFER, NO_SID, MISSING_SRV6_STATE, NAT46, NAT64, AUTH_REQUIRED, CT_NO_MAP_FOUND, SNAT_NO_MAP_FOUND, INVALID_CLUSTER_ID, UNSUPPORTED_PROTOCOL_FOR_DSR_ENCAP, NO_EGRESS_GATEWAY, UNENCRYPTED_TRAFFIC, TTL_EXCEEDED, NO_NODE_ID, DROP_RATE_LIMITED, IGMP_HANDLED, IGMP_SUBSCRIBED, MULTICAST_HANDLED, DROP_HOST_NOT_READY, DROP_EP_NOT_READY, DROP_NO_EGRESS_IP

- **spec.flowLogs.denyFilterList.event_type**
массив объектов

Типы событий для фильтрации.

- **spec.flowLogs.denyFilterList.event_type.match_sub_type**
булевый

Нужно ли сопоставлять также sub_type.

- **spec.flowLogs.denyFilterList.event_type.sub_type**
целочисленный

Вторичный тип события.

- **spec.flowLogs.denyFilterList.event_type.type**
целочисленный

Основной тип события.

- **spec.flowLogs.denyFilterList.http_header**
массив объектов

HTTP-заголовки для сопоставления.

- **spec.flowLogs.denyFilterList.http_header.key**
строка

Ключ заголовка.

- **spec.flowLogs.denyFilterList.http_header.value**
строка

Значение заголовка.

- **spec.flowLogs.denyFilterList.http_method**
массив строк

HTTP-методы.

- **Элемент массива**
строка

Допустимые значения: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH

- **spec.flowLogs.denyFilterList.http_path**
массив строк

Список регулярных выражений для сопоставления с HTTP-путем.

- **spec.flowLogs.denyFilterList.http_status_code**

массив строк

Префиксы HTTP-кодов ответа.

- **Элемент массива**

строка

Шаблон: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`

- **spec.flowLogs.denyFilterList.http_url**

массив строк

Список регулярных выражений для сопоставления с HTTP-адресом.

- **spec.flowLogs.denyFilterList.interface**

массив объектов

Сетевой интерфейс, на котором был замечен поток.

- **spec.flowLogs.denyFilterList.interface.index**

целочисленный

Индекс интерфейса.

Допустимые значения: `0 <= X`

- **spec.flowLogs.denyFilterList.interface.name**

строка

Имя интерфейса.

- **spec.flowLogs.denyFilterList.ip_version**

массив строк

Версия IP.

- **Элемент массива**

строка

Допустимые значения: `IP_NOT_USED`, `IPv4`, `IPv6`

- **spec.flowLogs.denyFilterList.node_labels**

массив строк

Селекторы лейблов узлов.

Примеры:

```
node_labels: node-role.kubernetes.io/control-plane=
```

```
node_labels: topology.kubernetes.io/ru-central1-a
```

- **spec.flowLogs.denyFilterList.node_name**

массив строк

Шаблоны имён узлов.

- **spec.flowLogs.denyFilterList.protocol**

массив строк

Протокол L4 или L7.

- **spec.flowLogs.denyFilterList.source_cluster_name**

массив строк

Имена кластеров источника.

- **spec.flowLogs.denyFilterList.source_fqdn**

массив строк

Полные доменные имена (FQDN) источника.

- **Элемент массива**

строка

Максимальная длина: `255`

- **spec.flowLogs.denyFilterList.source_identity**

массив целых чисел

Идентификатор безопасности (security identity) источника.

- **Элемент массива**

целочисленный

Допустимые значения: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.source_ip**

массив строк

IP-адрес или CIDR источника.

- **spec.flowLogs.denyFilterList.source_ip_xlated**

массив строк

IP-адрес или CIDR источника после трансляции (SNAT).

- **spec.flowLogs.denyFilterList.source_label**

массив строк

Селекторы лейблов источника.

Для получения дополнительной информации смотрите документацию Cilium.

Примеры:

```
source_label: k8s:role=frontend
```

```
source_label: reserved:kube-apiserver
```

- **spec.flowLogs.denyFilterList.source_pod**

массив строк

Префиксы имён подов источника в формате NamespacedName (namespace/pod).

Примеры:

```
source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
```

```
source_pod: kube-system/
```

```
source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
```

- **spec.flowLogs.denyFilterList.source_port**

массив строк

Порт L4 источника.

- **Элемент массива**

строка

Шаблон: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.source_service**

массив строк

Имена сервисов источника в формате NamespacedName (namespace/service).

Примеры:

```
source_service: kube-system/kube-dns
```

```
source_service: kube-system/
```

```
source_service: "/kube-dns"
```

- **spec.flowLogs.denyFilterList.source_workload**

массив объектов

Нагрузки источника.

- **spec.flowLogs.denyFilterList.source_workload.kind**

строка

Тип нагрузки.

- **spec.flowLogs.denyFilterList.source_workload.name**

строка

Имя нагрузки.

- **spec.flowLogs.denyFilterList.tcp_flags**

массив объектов

TCP-флаги для сопоставления.

- **spec.flowLogs.denyFilterList.tcp_flags.ACK**

- булевый
- **spec.flowLogs.denyFilterList.tcp_flags.CWR**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.ECE**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.FIN**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.NS**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.PSH**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.RST**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.SYN**
булевый
- **spec.flowLogs.denyFilterList.tcp_flags.URG**
булевый
- **spec.flowLogs.denyFilterList.trace_id**
массив строк
ID трассировки.
- **spec.flowLogs.denyFilterList.traffic_direction**
массив строк
Направление трафика.
 - **Элемент массива**
строка
Допустимые значения: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`
- **spec.flowLogs.denyFilterList.uuid**
массив строк
Фильтрация по UUID потоков (flow UUID).
- **spec.flowLogs.denyFilterList.verdict**
массив строк
Сопоставление по вердикту.
 - **Элемент массива**
строка
Допустимые значения: `VERDICT_UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`
- **spec.flowLogs.enabled**
булевый
Включает ведение журналов событий для агентов Cilium.
Файл журнала расположен на файловой системе узла по пути `/var/log/cilium/hubble/flow.log`.
По умолчанию: `false`
- **spec.flowLogs.fieldMaskList**
массив строк
Список полей, которые должны отображаться в событиях.
Для получения дополнительной информации смотрите документацию Cilium.
 - **Элемент массива**
строка
Допустимые значения: `time`, `uuid`, `verdict`, `drop reason_desc`, `auth type`, `ethernet`, `ethernet.source`, `ethernet.destination`, `IP`, `IP.source`, `IP.source xlated`, `IP.destination`, `IP.ipVersion`, `IP.encrypted`, `l4`, `l4.TCP`, `l4.TCP.source port`, `l4.TCP.destination port`, `l4.TCP.flags`, `l4.TCP.flags.FIN`, `l4.TCP.flags.SYN`, `l4.TCP.flags.RST`, `l4.TCP.flags.PSH`, `l4.TCP.flags.ACK`, `l4.TCP.flags.URG`, `l4.TCP.flags.ECE`, `l4.TCP.flags.CWR`, `l4.TCP.flags.NS`, `l4.UDP`, `l4.UDP.source port`, `l4.UDP.destination port`, `l4.ICMPv4`, `l4.ICMPv4.type`, `l4.ICMPv4.code`, `l4.ICMPv6`, `l4.ICMPv6.type`, `l4.ICMPv6.code`, `l4.SCTP`, `l4.SCTP.source port`, `l4.SCTP.destination port`, `source`, `source.ID`, `source.identity`, `source.cluster_name`, `source.namespace`, `source.labels`, `source.pod name`, `source.workloads`, `source.workloads.name`, `source.workloads.kind`, `destination`, `destination.ID`, `destination.identity`, `destination.cluster_name`, `destination.namespace`, `destination.labels`, `destination.pod name`, `destination.workloads`, `destination.workloads.name`, `destination.workloads.kind`, `Type`, `node name`, `node labels`, `source names`, `destination names`, `l7`, `l7.type`, `l7.latency_ns`, `l7.record`, `l7.record.dns`, `l7.record.dns.query`, `l7.record.dns.ips`, `l7.record.dns.ttl`, `l7.record.dns.cnames`, `l7.record.dns.observation source`, `l7.record.dns.rcode`, `l7.record.dns.qtypes`, `l7.record.dns.rtypes`, `l7.record.http`, `l7.record.http.code`, `l7.record.http.method`, `l7.record.http.url`,

```
l7.record.http.protocol, l7.record.http.headers, l7.record.http.headers.key,
l7.record.http.headers.value, l7.record.kafka, l7.record.kafka.error_code,
l7.record.kafka.api version, l7.record.kafka.api key,
l7.record.kafka.correlation id, l7.record.kafka.topic, is reply, event type,
event type.type, event type.sub type, traffic direction, trace observation_point,
trace reason, file, file.name, file.line, debug capture point, interface,
interface.index, interface.name, proxy port, trace context, trace context.parent,
trace context.parent.trace id, sock xlate point, socket_cookie, cgroup_id,
extensions, egress allowed by, egress allowed by.name,
egress allowed by.namespace, egress allowed by.labels, egress allowed_by.revision,
egress allowed by.kind, ingress allowed by, ingress allowed_by.name,
ingress allowed by.namespace, ingress allowed by.labels,
ingress allowed by.revision, ingress allowed by.kind, egress denied by,
egress denied by.name, egress denied by.namespace, egress denied by.labels,
egress denied by.revision, egress denied by.kind, ingress denied by,
ingress denied by.name, ingress denied by.namespace, ingress denied_by.labels,
ingress denied by.revision, ingress denied by.kind, source service,
source service.name, source service.namespace, destination_service,
destination_service.name, destination_service.namespace
```

- **spec.flowLogs.fileMaxSizeMB**
целочисленный

Размер в МБ, при достижении которого будет выполняться ротация файла экспорта Hubble.

По умолчанию: 10

Допустимые значения: `1 <= X <= 150`

Примеры

Egress Gateway

Принцип работы

Для настройки egress-шлюза необходимы два кастомных ресурса:

1. EgressGateway — описывает группу узлов, один из которых будет выбран в качестве активного egress-шлюза, а остальные останутся в резерве на случай отказа:
 - Среди группы узлов, попадающих под `spec.nodeSelector`, будут отобраны пригодные к использованию. Один из них будет назначен активным шлюзом. Выбор активного узла осуществляется в алфавитном порядке.

Признаки пригодного узла:

- Узел в состоянии `Ready`.
 - Узел не находится в состоянии технического обслуживания (`cordon`).
 - `cilium-agent` на узле в состоянии `Ready`.
 - При использовании EgressGateway в режиме `VirtualIP` на активном узле запускается агент, который эмулирует «виртуальный» IP-адрес с использованием протокола ARP. При определении пригодности узла также учитывается состояние пода данного агента.
 - Разные EgressGateway могут использовать одни и те же узлы. Выбор активного узла в каждом EgressGateway осуществляется независимо от других, что позволяет сбалансировать нагрузку между ними.
2. EgressGatewayPolicy — описывает политику перенаправления сетевых запросов от подов в кластере на конкретный egress-шлюз, определённый с помощью EgressGateway.

Обслуживание узла

Для проведения работ на узле, который в данный момент является активным egress-шлюзом, выполните следующие шаги:

1. Снимите лейбл с узла, чтобы исключить его из списка кандидатов для роли egress-шлюза. Egress-label — это лейбл, указанный в `spec.nodeSelector` вашего EgressGateway.

```
d8 k label node <node-name> <egress-label>-
```

2. Переведите узел в режим обслуживания (`cordon`), чтобы предотвратить запуск новых подов:

```
d8 k cordon <node-name>
```

После этого Cilium автоматически выберет новый активный узел из оставшихся кандидатов. Трафик продолжит направляться через новый шлюз без прерывания.

3. Для возврата узла в работу выполните:

```
d8 k uncordon <node-name>
d8 k label node <node-name> <egress-label>=<value>
```

i Важно: повторное добавление лейбла может привести к тому, что узел снова будет выбран активным egress-шлюзом (если он первый в алфавитном порядке среди доступных кандидатов). Чтобы избежать немедленного возврата узла в активное состояние, временно уменьшите количество реплик в EgressGateway или настройте приоритет выбора через дополнительные лейблы.

Сравнение с CiliumEgressGatewayPolicy

CiliumEgressGatewayPolicy подразумевает настройку только одного узла в качестве egress-шлюза. При выходе его из строя не предусмотрено failover-механизмов и сетевая связь будет нарушена.

Примеры настроек Egress Gateway

EgressGateway в режиме PrimaryIPFromEgressGatewayNodeInterface (базовый режим)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: myegressgw
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # На всех узлах, попадающих под nodeSelector, «публичный» интерфейс должен называться
      # одинаково.
      # При выходе из строя активного узла, трафик будет перенаправлен через резервный и
      # IP-адрес отправителя у сетевых пакетов поменяется.
      interfaceName: eth1
```

EgressGateway в режиме VirtualIPAddress (режим с Virtual IP)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: myeg
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # На каждом узле должны быть настроены все необходимые маршруты для доступа на все внешние
      # публичные сервисы,
      # «публичный» интерфейс должен быть подготовлен к автоматической настройке «виртуального» IP
      # в качестве дополнительного (secondary) IP-адреса.
      # При выходе из строя активного узла, трафик будет перенаправлен через резервный и
      # IP-адрес отправителя у сетевых пакетов не поменяется.
      ip: 172.18.18.242
      # Список сетевых интерфейсов для «виртуального» IP.
      interfaces:
        - eth1
```

EgressGatewayPolicy


```

apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
    - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
    - podSelector:
        matchLabels:
          app: backend
          io.kubernetes.pod.namespace: my-ns

```

HubbleMonitoringConfig

Кластерный ресурс HubbleMonitoringConfig предназначен для настройки экспорта данных из Hubble, работающего внутри агентов Cilium.

Примеры настроек HubbleMonitoringConfig

Включение расширенных метрик и экспорта flow logs (с фильтрами и маской полей)



Ресурс HubbleMonitoringConfig **должен иметь имя** hubble-monitoring-config.

Пример включения метрик и экспорта:

```

apiVersion: deckhouse.io/v1alpha1
kind: HubbleMonitoringConfig
metadata:
  name: hubble-monitoring-config
spec:
  extendedMetrics:
    enabled: true
    collectors:
      - name: drop
        # Добавить дополнительный контекст (лейблы) для выбранного коллектора.
        contextOptions:
          "labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,destination_pod"
      - name: flow
  flowLogs:
    enabled: true
    # Записывать в лог-файл /var/log/cilium/hubble/flow.log только указанные события.
    allowFilterList:
      - verdict:
          - DROPPED
          - ERROR
    # Исключить из лог-файла события, соответствующие denyFilterList.
    denyFilterList:
      - source_pod:
          - kube-system/
      - destination_pod:
          - kube-system/
    # Сохранять в каждой записи только указанные поля.
    fieldMaskList:
      - time
      - verdict
    # Максимальный размер лог-файла (в МБ) перед ротацией.
    fileMaxSizeMB: 30

```

Сбор Hubble flow logs с помощью модуля log-shipper

Для сбора flow logs используйте модуль `log-shipper`.

Создайте ресурс ClusterLoggingConfig, который читает лог-файл с файловой системы узла:

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: cilium-hubble-flow-logs
spec:
  type: File
  file:
    include:
      - /var/log/cilium/hubble/flow.log

```

Модуль cni-flannel

Обеспечивает работу сети в кластере с помощью модуля flannel.

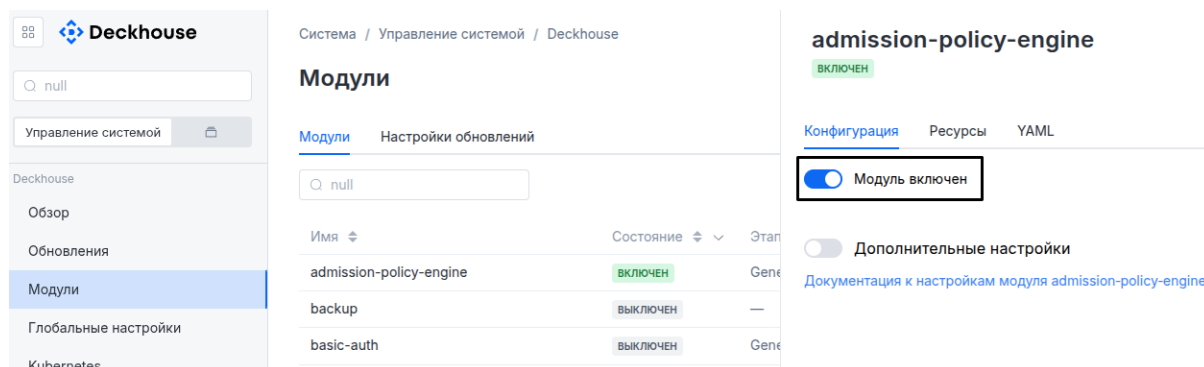
Настройки

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-flannel`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `cni-flannel`:

```
d8 system module enable cni-flannel
```

- **С помощью ModuleConfig `cni-flannel`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `cni-flannel` (создайте его, при необходимости).

Пример манифеста для включения модуля `cni-flannel`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-flannel
spec:
  enabled: true

```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-flannel`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация

Ресурсы

YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация

YAML

Вы также можете отредактировать объект ModuleConfig `cni-flannel` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `cni-flannel`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `cni-flannel` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc cni-flannel
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `cni-flannel`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `cni-flannel`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-flannel
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.podNetworkMode**
строка

Режим работы.

i

Внимание. При переключении между `HostGW` и `VXLAN` требуется **перезагрузка всех узлов кластера**. Изменять этот параметр можно **только** при использовании модуля в **bare-metal-кластерах**.

По умолчанию: `HostGW`

Допустимые значения: `HostGW`, `VXLAN`

Модуль cni-simple-bridge

Модуль не имеет настроек.

Включается автоматически для следующих облачных провайдеров:

- AWS.
- Azure.
- GCP.

Управление control plane

Управление компонентами control plane кластера осуществляется с помощью модуля `control-plane-manager`, который запускается на всех master-узлах кластера (узлы с лейблом `node-role.kubernetes.io/control-plane: ""`).

Функции управления control plane:

- **Управление сертификатами**, необходимыми для работы control-plane, в том числе продление, выпуск при изменении конфигурации и т. п. Позволяет автоматически поддерживать безопасную конфигурацию control plane и быстро добавлять дополнительные SAN для организации защищенного доступа к API Kubernetes.
- **Настройка компонентов**. Автоматически создает необходимые конфигурации и манифесты компонентов `control-plane`.
- **Upgrade/downgrade компонентов**. Поддерживает в кластере одинаковые версии компонентов.
- **Управление конфигурацией etcd-кластера** и его членов. Масштабирует master-узлы, выполняет миграцию из single-master в multi-master и обратно.
- **Настройка kubeconfig**. Обеспечивает актуальные файлы kubeconfig на узлах control-plane. Генерирует, продлевает и обновляет kubeconfig для компонентов control-plane и admin kubeconfig (`admin.conf`). По умолчанию создаёт символическую ссылку для root-пользователя (`/root/.kube/config -> admin.conf`). При включённом модуле `user-authz` симлинк можно отключить параметром `rootKubeconfigSymlink` в модуле **control-plane-manager** (см. FAQ). Также ужесточает права доступа к файлам `admin.conf` и `super-admin.conf`.
- **Расширение работы планировщика**, за счет подключения внешних плагинов через вебхуки. Управляется ресурсом `KubeSchedulerWebhookConfiguration`. Позволяет использовать более сложную логику при решении задач планирования нагрузки в кластере. Например:
 - размещение подов приложений организации хранилища данных ближе к самим данным,
 - приоритизация узлов в зависимости от их состояния (сетевой нагрузки, состояния подсистемы хранения и т. д.),
 - разделение узлов на зоны, и т. п.

Управление сертификатами

Управляет SSL-сертификатами компонентов `control-plane`:

- Серверными сертификатами для `kube-apiserver` и `etcd`. Они хранятся в Secret'e `d8-pki` пространства имен `kube-system`:
 - корневой CA `kubernetes` (`ca.crt` и `ca.key`);
 - корневой CA `etcd` (`etcd/ca.crt` и `etcd/ca.key`);
 - RSA-сертификат и ключ для подписи Service Account'ов (`sa.pub` и `sa.key`);
 - корневой CA для extension API-серверов (`front-proxy-ca.key` и `front-proxy-ca.crt`).
- Клиентскими сертификатами для подключения компонентов `control-plane` друг к другу. Выписывает, продлевает и перевыписывает, если что-то изменилось (например, список SAN). Следующие сертификаты хранятся только на узлах:
 - серверный сертификат API-сервера (`apiserver.crt` и `apiserver.key`);
 - клиентский сертификат для подключения `kube-apiserver` к `kubelet` (`apiserver-kubelet-client.crt` и `apiserver-kubelet-client.key`);
 - клиентский сертификат для подключения `kube-apiserver` к `etcd` (`apiserver-etcd-client.crt` и `apiserver-etcd-client.key`);
 - клиентский сертификат для подключения `kube-apiserver` к extension API-серверам (`front-proxy-client.crt` и `front-proxy-client.key`);
 - серверный сертификат `etcd` (`etcd/server.crt` и `etcd/server.key`);
 - клиентский сертификат для подключения `etcd` к другим членам кластера (`etcd/peer.crt` и `etcd/peer.key`);
 - клиентский сертификат для подключения `kubelet` к `etcd` для healthcheck'ов (`etcd/healthcheck-client.crt` и `etcd/healthcheck-client.key`).

Также позволяет добавить дополнительные SAN в сертификаты, это дает возможность быстро и просто добавлять дополнительные «точки входа» в API Kubernetes.

При изменении сертификатов также автоматически обновляется соответствующая конфигурация `kubeconfig`.

Масштабирование

Поддерживается работа `control-plane` в конфигурации как *single-master*, так и *multi-master*.

В конфигурации *single-master*:

- `kube-apiserver` использует только тот экземпляр `etcd`, который размещен с ним на одном узле;
- На узле настраивается прокси-сервер, отвечающий на localhost, `kube-apiserver` отвечает на IP-адрес master-узла.

В конфигурации *multi-master* компоненты `control-plane` автоматически разворачиваются в отказоустойчивом режиме:

- `kube-apiserver` настраивается для работы со всеми экземплярами `etcd`.
- На каждом master-узле настраивается дополнительный прокси-сервер, отвечающий на localhost. Прокси-сервер по умолчанию обращается к локальному экземпляру `kube-apiserver`, но в случае его

недоступности последовательно опрашивает остальные экземпляры `kube-apiserver`.

Масштабирование master-узлов

Масштабирование узлов `control-plane` осуществляется автоматически, с помощью лейбла `node-role.kubernetes.io/control-plane=""`:

- Установка лейбла `node-role.kubernetes.io/control-plane=""` на узле приводит к разворачиванию на нем компонентов `control-plane`, подключению нового узла `etcd` в `etcd`-кластер, а также регенерации необходимых сертификатов и конфигурационных файлов.
- Удаление лейбла `node-role.kubernetes.io/control-plane=""` с узла приводит к удалению всех компонентов `control-plane`, регенерации необходимых конфигурационных файлов и сертификатов, а также корректному исключению узла из `etcd`-кластера.



При масштабировании узлов с 2 до 1 требуются ручные действия с `etcd`. В остальных случаях все необходимые действия происходят автоматически. Обратите внимание, что при масштабировании с любого количества master-узлов до 1 рано или поздно на последнем шаге возникнет ситуация масштабирования узлов с 2 до 1.

Динамическое пороговое значение удаления выселенных подов

Автоматически настраивает оптимальное значение `--terminated-pod-gc-threshold` в зависимости от размера кластера:

- **Малые кластеры** (менее 100 узлов): 1000 завершенных подов.
- **Средние кластеры** (от 100 до 300 узлов): 3000 завершенных подов.
- **Крупные кластеры** (от 300 узлов): 6000 завершенных подов.



Эта функция применяется только в средах, где параметр `--terminated-pod-gc-threshold` можно настраивать. В управляемых Kubernetes-кластерах, таких как EKS, GKE, AKS, это значение контролируется провайдером.

Управление версиями

Обновление **patch-версии** компонентов control plane (то есть в рамках минорной версии, например с `1.31.13` на `1.31.14`) происходит автоматически вместе с обновлением версии Deckhouse. Управлять обновлением patch-версий нельзя.

Обновлением **минорной-версии** компонентов control plane (например, с `1.31.*` на `1.32.*`) можно управлять с помощью параметра `kubernetesVersion`, в котором можно выбрать автоматический режим обновления (значение `Automatic`) или указать желаемую минорную версию control plane. Версию control plane, которая используется по умолчанию (при `kubernetesVersion: Automatic`), а также список поддерживаемых версий Kubernetes можно найти в документации.

Обновление control plane выполняется безопасно и для single-master-, и для multi-master-кластеров. Во время обновления может быть кратковременная недоступность API-сервера. На работу приложений в кластере обновление не влияет и может выполняться без выделения окна для регламентных работ.

Если указанная для обновления версия (с параметром `kubernetesVersion`) не соответствует текущей версии control plane в кластере, запускается умная стратегия изменения версий компонентов:

- Общие замечания:
 - Обновление в разных NodeGroup выполняется параллельно. Внутри каждой NodeGroup узлы обновляются последовательно, по одному.
- При upgrade:
 - Обновление происходит **последовательными этапами**, по одной минорной версии: `1.31 -> 1.32`, `1.32 -> 1.33`, `1.33 -> 1.34`.
 - На каждом этапе сначала обновляется версия control plane, затем происходит обновление kubelet на узлах кластера.
- При downgrade (не поддерживается для редакций CSE):
 - Успешное понижение версии гарантируется только на одну версию вниз от максимальной минорной версии control plane, когда-либо использовавшейся в кластере.
 - Сначала на узлах кластера выполняется понижение версии kubelet, после чего производится понижение версии компонентов control plane.

Аудит

Если требуется журналировать операции с API или отдебажить неожиданное поведение, для этого в Kubernetes предусмотрен Auditing. Его можно настроить путем создания правил Audit Policy, а результатом работы аудита будет лог-файл `/var/log/kube-audit/audit.log` со всеми интересующими операциями.

В установках Deckhouse по умолчанию созданы базовые политики, которые отвечают за логирование событий, которые:

- связаны с операциями создания, удаления и изменения ресурсов;
- совершаются от имен сервисных аккаунтов из системных Namespace `kube-system`, `d8-*`;
- совершаются с ресурсами в системных пространствах имен `kube-system`, `d8-*`.

Для выключения базовых политик установите флаг `basicAuditPolicyEnabled` в `false`.

При настройке OIDC-аутентификации в аудит-логах дополнительно включается информация о пользователе в поле `user.extra`:

- `user-authn.deckhouse.io/name` — отображаемое имя пользователя
- `user-authn.deckhouse.io/preferred-username` — предпочитаемое имя пользователя
- `user-authn.deckhouse.io/dex-provider` — идентификатор провайдера Dex (требуется `scope federated:id`)

Настройка политик аудита подробнее рассмотрена в одноименной секции FAQ.

Feature Gates

Управление feature gates осуществляется с помощью параметра `enabledFeatureGates` ModuleConfig `control-plane-manager`.

Изменение списка feature gates вызывает перезапуск соответствующего компонента (например, `kube-apiserver`, `kube-scheduler`, `kube-controller-manager`, `kubelet`).

Пример включения feature gates `ComponentFlagz` и `ComponentStatusz`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    enabledFeatureGates:
      - ComponentFlagz
      - ComponentStatusz
```

Если feature gate не поддерживается или имеет статус `deprecated`, в системе мониторинга будет сгенерирован алерт `D8ProblematicFeatureGateInUse`, информирующий о том, что feature gate не будет применен.



Обновление версии Kubernetes (управляется параметром `kubernetesVersion`) не произойдет, если в списке включенных feature gates, заданных для новой версии Kubernetes, есть feature gates в статусе `deprecated`.

Описание feature gates доступно в документации Kubernetes.

▼ Список доступных feature gates...

Kubernetes 1.31

- `AuthorizeNodeWithSelectors`
- `AuthorizeWithSelectors`
- `ConcurrentWatchObjectDecode`
- `CrossNamespaceVolumeDataSource`
- `DisableAllocatorDualWrite`
- `HPAScaleToZero`
- `ImageVolume`
- `InPlacePodVerticalScaling`
- `JobManagedBy`
- `KubeletPodResourcesGet`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `OrderedNamespaceDeletion`
- `ProcMountType`
- `RecoverVolumeExpansionFailure`
- `RelaxedEnvironmentVariableValidation`
- `ResourceHealthStatus`
- `SELinuxMount`
- `SchedulerQueueingHints`
- `StrictCostEnforcementForVAP`
- `StrictCostEnforcementForWebhooks`

Kubernetes 1.32

- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `CrossNamespaceVolumeDataSource`

- DisableAllocatorDualWrite
- HPAScaleToZero
- ImageVolume
- InPlacePodVerticalScaling
- InPlacePodVerticalScalingAllocatedStatus
- KubeletPodResourcesGet
- MaxUnavailableStatefulSet
- MemoryQoS
- MutatingAdmissionPolicy
- OrderedNamespaceDeletion
- PodLifecycleSleepActionAllowZero
- PodLogsQuerySplitStreams
- ProcMountType
- RelaxedDNSSearchValidation
- ResourceHealthStatus
- SELinuxChangePolicy
- SELinuxMount
- SchedulerAsyncPreemption

Kubernetes 1.33

- AllowParsingUserUIDFromCertAuth
- AllowUnsafeMalformedObjectDeletion
- ComponentFlagz
- ComponentStatusz
- ConcurrentWatchObjectDecode
- ContainerStopSignals
- CrossNamespaceVolumeDataSource
- DeploymentReplicaSetTerminatingReplicas
- DisableAllocatorDualWrite
- HPAConfigurableTolerance
- HPAScaleToZero
- ImageVolume
- KubeletEnsureSecretPulledImages
- KubeletPSI
- KubeletPodResourcesGet
- ListFromCacheSnapshot
- MaxUnavailableStatefulSet
- MemoryQoS
- MutatingAdmissionPolicy
- PodLogsQuerySplitStreams
- PodObservedGenerationTracking
- PreferSameTrafficDistribution
- ResourceHealthStatus
- SELinuxMount
- StrictIPCIDRValidation

Kubernetes 1.34

- AllowUnsafeMalformedObjectDeletion
- ComponentFlagz
- ComponentStatusz
- ConcurrentWatchObjectDecode
- ContainerRestartRules
- ContainerStopSignals
- CrossNamespaceVolumeDataSource
- DeploymentReplicaSetTerminatingReplicas
- EnvFiles
- HPAConfigurableTolerance
- HPAScaleToZero
- HostnameOverride
- ImageVolume
- KubeletEnsureSecretPulledImages
- MaxUnavailableStatefulSet
- MemoryQoS
- MutatingAdmissionPolicy
- PodLogsQuerySplitStreams
- ResourceHealthStatus
- SELinuxMount
- StrictIPCIDRValidation

Kubernetes 1.35

- AllowUnsafeMalformedObjectDeletion
- CRDObservedGenerationTracking
- ComponentFlagz
- ComponentStatusz
- ConcurrentWatchObjectDecode
- ConstrainedImpersonation
- ContainerStopSignals
- CrossNamespaceVolumeDataSource

- `HPAScaleToZero`
- `MemoryQoS`
- `MutablePVNodeAffinity`
- `MutablePodResourcesForSuspendedJobs`
- `MutableSchedulingDirectivesForSuspendedJobs`
- `MutatingAdmissionPolicy`
- `NodeDeclaredFeatures`
- `PodLogsQuerySplitStreams`
- `ResourceHealthStatus`
- `RestartAllContainersOnContainerExits`
- `SELinuxMount`
- `StrictIPCIDRValidation`
- `TaintTolerationComparisonOperators`

Защита чувствительных полей кастомных ресурсов

Feature gate `CRDSensitiveData` обеспечивает защиту чувствительных данных на уровне полей в ресурсах, помеченных маркером `x-kubernetes-sensitive-data: true`. Функция реализована в виде патча к `kube-apiserver` (`apiextensions-apiserver`) и поддерживается, начиная с версии Kubernetes 1.31.

Маркер `x-kubernetes-sensitive-data` проверяется `kube-apiserver` при применении ресурса:

- маркер требует, чтобы был включен feature gate `CRDSensitiveData` (включается автоматически, когда `apiserver.encryptionEnabled` равен `true`);
- маркер не допускается устанавливать на корне схемы (на самом узле `openAPIV3Schema`). Чтобы защитить все поля ресурса, добавьте маркер на свойство `spec` (или на поддерево ниже него), а не на корень схемы — на корне также находятся системные поля (`apiVersion`, `kind`, `metadata`), которые невозможно зашифровать;
- тип поля должен быть одним из типов OpenAPI v3: `string`, `integer`, `number`, `boolean`, `object` или `array`. Маркер на `object` или `array` делает чувствительным всё поддерево;
- поддерживаются поля, объявленные как `x-kubernetes-int-or-string: true`;
- маркер запрещён внутри веток `anyOf`, `oneOf`, `allOf` и `not` (это проверяет валидатор структурной схемы).

Если хотя бы одно поле в схеме ресурса помечено `x-kubernetes-sensitive-data: true`, ко всем кастомным ресурсам этого типа применяются следующие меры защиты:

- **Шифрование в etcd** — весь ресурс шифруется с помощью того же механизма, что и Kubernetes Secrets. Требуется включения параметра `apiserver.encryptionEnabled`.
- **Фильтрация полей на основе RBAC** — при выполнении запросов `get`, `list`, или `watch` чувствительные поля удаляются из ответа API, если у вызывающей стороны нет прав `get`, `list` или `watch` на субресурс `<resource>/sensitive`.
- **Маскировка в журнале аудита** — значения чувствительных полей всегда заменяются на `"*****"` в журнале аудита, независимо от прав RBAC и уровня аудита.

Чтобы включить защиту чувствительных полей, установите параметр `apiserver.encryptionEnabled` в `true`. Feature gate `CRDSensitiveData` включается автоматически при активации шифрования, его не следует указывать вручную:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true
```



Включение `encryptionEnabled` необратимо и приводит к перезапуску `kube-apiserver`.

За подробностями обратитесь к следующим разделам:

- «FAQ» — инструкция по включению защиты чувствительных полей;
- «Примеры» — примеры конфигурации и результатов.

Настройки

Некоторые параметры кластера, влияющие на управление control plane, также берутся из ресурса `ClusterConfiguration`.

В модуле реализовано 10 алертов.

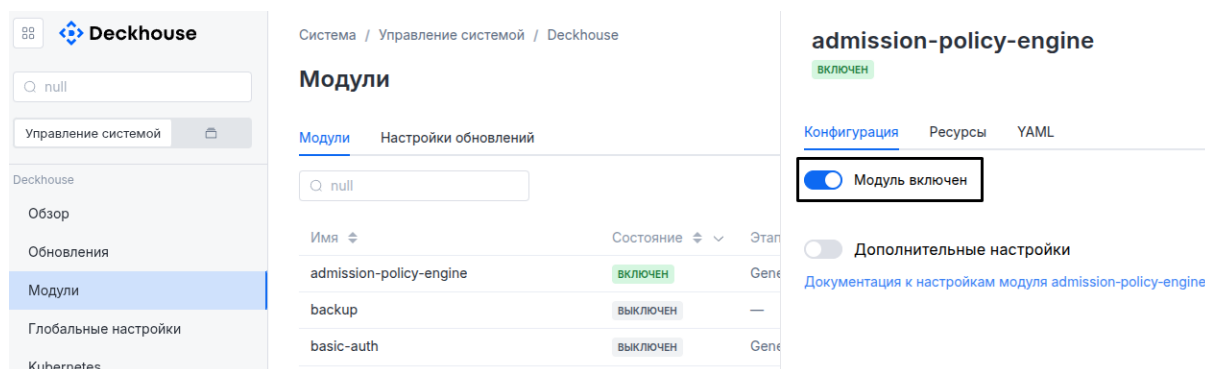
Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» →

«Deckhouse» → «Модули», откройте модуль `control-plane-manager`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `control-plane-manager`:

```
d8 system module enable control-plane-manager
```

- **С помощью ModuleConfig `control-plane-manager`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `control-plane-manager` (создайте его, при необходимости).

Пример манифеста для включения модуля `control-plane-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
```

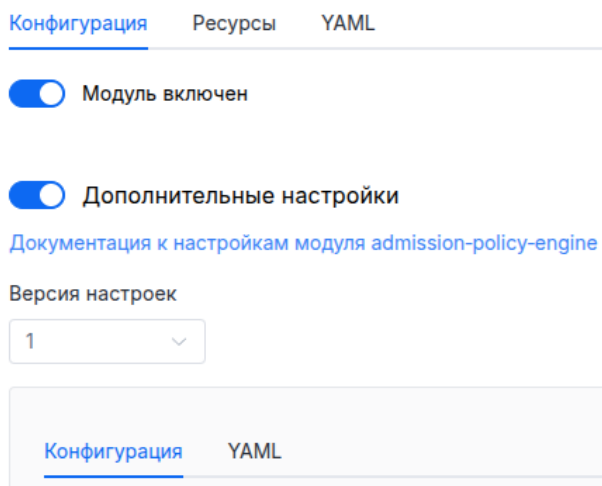
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `control-plane-manager`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `control-plane-manager` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `control-plane-manager`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `control-plane-manager` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc control-plane-manager
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `control-plane-manager`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `control-plane-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Удалите `etcd.externalMembersNames`

Параметры

Версия схемы: 2

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
        - bakery.infra
        - devs.infra
      loadBalancer: {}
```

- **settings**

объект

- **settings.apiserver**
объект

Параметры `kube-apiserver`.

- **settings.apiserver.admissionPlugins**
массив строк

Список включенных дополнительных admission-плагинов.

Обратите внимание, что, помимо включенных по умолчанию в Kubernetes admission-плагинов, также всегда включены (без возможности отключения) следующие admission-плагины:

- `ExtendedResourceToleration`;

- `PodNodeSelector`;
- `PodTolerationRestriction`;
- `EventRateLimit`, который использует следующую конфигурацию:

```
apiVersion: eventratelimit.admission.k8s.io/v1alpha1
kind: Configuration
limits:
- type: Namespace
  qps: 50
  burst: 100
  cacheSize: 2000
```

i Admission-плагин `PodNodeSelector` не требует указания глобальной конфигурации, он использует только аннотации на пространстве имен (namespace'ах).

Пример:

```
admissionPlugins:
- AlwaysPullImages
- NamespaceAutoProvision
```

- **Элемент массива**
строка

Допустимые значения: `AlwaysPullImages`, `NamespaceAutoProvision`, `OwnerReferencesPermissionEnforcement`, `PodNodeSelector`, `PodTolerationRestriction`

- **`settings.apiserver.auditLog`**
объект

Параметры аудита.

По умолчанию: `{"output": "File"}`

- **`settings.apiserver.auditLog.output`**
строка

Обязательный параметр

Тип потока для вывода журнала аудита.

По умолчанию: `File`

Допустимые значения: `File`, `Stdout`

Пример:

```
output: Stdout
```

- **`settings.apiserver.auditLog.path`**
строка

Путь к директории. Действует только в случае `output: "File"`.

По умолчанию: `/var/log/kube-audit`

Шаблон: `^[a-zA-Z0-9_/.-]+[a-zA-Z0-9_-.]$\`

- **`settings.apiserver.auditPolicyEnabled`**
булевый

Применяет дополнительные политики аудита с конфигурацией из Secret'a `kube-system/audit-policy`.

По умолчанию: `false`

- **`settings.apiserver.authn`**
объект

Опциональные параметры аутентификации клиентов Kubernetes API.

По умолчанию используются данные из ConfigMap, устанавливаемого модулем `user-authn`.

- **`settings.apiserver.authn.oidcCA`**
строка

CA OIDC-провайдера.

- **`settings.apiserver.authn.oidcIssuerAddress`**
строка

Сетевой адрес OIDC-провайдера.

Примеры:

```
oidcIssuerAddress: 1.2.3.4
```

```
oidcIssuerAddress: ''
```

■ **settings.apiserver.authn.oidcIssuerURL**

строка

URL OIDC-провайдера.

Пример:

```
oidcIssuerURL: https://my-super-site.tech/
```

■ **settings.apiserver.authn.webhookCA**

строка

CA аутентификационного webhook'а.

■ **settings.apiserver.authn.webhookCacheTTL**

строка

Продолжительность кеширования ответов от аутентификационного webhook'а.

Задается в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

Шаблон: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

Пример:

```
webhookCacheTTL: 5m
```

■ **settings.apiserver.authn.webhookURL**

строка

URL аутентификационного webhook'а.

Пример:

```
webhookURL: https://127.0.0.1:40443/
```

■ **settings.apiserver.authz**

объект

Параметры авторизации клиентов Kubernetes API.

По умолчанию используются данные из ConfigMap, устанавливаемого модулем user-authz.

■ **settings.apiserver.authz.webhookCA**

строка

CA webhook'а авторизации.

■ **settings.apiserver.authz.webhookURL**

строка

URL webhook'а авторизации.

Пример:

```
webhookURL: https://127.0.0.1:40443/
```

■ **settings.apiserver.basicAuditPolicyEnabled**

булевый

Применяет базовые политики аудита Deckhouse.

По умолчанию: `true`

■ **settings.apiserver.bindToWildcard**

булевый

Прослушивание на `0.0.0.0`.

По умолчанию API-сервер слушает на hostIP, который обычно соответствует Internal-адресу узла, но это зависит от типа кластера (Static или Cloud) и выбранной схемы размещения (layout).

По умолчанию: `false`

■ **settings.apiserver.certSANs**

массив строк

Массив строк, список дополнительных SANs, с которыми будет сгенерирован сертификат API-сервера.

Кроме переданного списка, всегда используется и следующий список:

- `kubernetes;`
- `kubernetes.default;`
- `kubernetes.default.svc;`
- `kubernetes.default.svc.cluster.local;`
- Адрес API-сервера кластера;
- `127.0.0.1;`
- DNS-имя хоста (hostname);
- IP-адрес хоста.

Пример:

```
certSANs:
- my-site.com
- 192.168.67.76
```

- **Элемент массива**
строка

Шаблон: `^[0-9a-zA-Z\.-]+\.`

- **settings.apiserver.encryptionEnabled**
булевый

Включает шифрование данных в etcd (Encrypting secret data at rest).

При включении создаётся Secret `kube-system/d8-secret-encryption-key` с ключом шифрования.

Включение этого параметра автоматически активирует feature gate `CRDSensitiveData` для `kube-apiserver`, который защищает поля кастомных ресурсов, помеченные маркером `x-kubernetes-sensitive-data: true`, через шифрование в etcd, фильтрацию полей через субресурс `<resource>/sensitive` на основе RBAC и маскировку значений в журнале аудита.

i

Внимание. После включения этот параметр нельзя отключить.

По умолчанию: `false`

- **settings.apiserver.loadBalancer**
объект

Создание сервиса `kube-system/d8-control-plane-apiserver` с типом `LoadBalancer`.

- **settings.apiserver.loadBalancer.annotations**
объект

Аннотации, которые будут проставлены сервису для гибкой настройки балансировщика.

i

Внимание. Модуль не учитывает особенности указания аннотаций в различных облаках. Если аннотации для заказа `LoadBalancer`'а применяются только при создании сервиса, для обновления подобных параметров вам необходимо будет удалить и добавить параметр `apiserver.loadBalancer`.

- **settings.apiserver.loadBalancer.port**
целочисленный

Внешний порт TCP-балансировщика.

По умолчанию: `443`

Допустимые значения: `1 <= X <= 65534`

- **settings.apiserver.loadBalancer.sourceRanges**
массив строк

Список CIDR, которым разрешен доступ к API.

Облачный провайдер может не поддерживать данную опцию и игнорировать ее.

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}`

- **settings.apiserver.serviceAccount**
объект

Настройки выпуска `ServiceAccount`'ов.

Примеры:

```
serviceAccount: {}
```

```
serviceAccount:
  additionalAPIAudiences:
  - istio-ca
```

- **settings.apiserver.serviceAccount.additionalAPIAudiences**

массив строк

Список дополнительных API audience'ов, которые следует добавить при создании токенов ServiceAccount.

Список audience'ов по умолчанию содержит:

- `serviceAccount.issuer`, если указано
- `serviceAccount.additionalAPIIssuers`, если указано
- `https://kubernetes.default.svc.${clusterDomain}`, например, `https://kubernetes.default.svc.cluster.local` - этот audience добавляется всегда для обратной совместимости.

Аутентификатор токенов ServiceAccount проверяет, что токены, использованные с API, привязаны как минимум к одной из этих audiences.

Этот вариант полезен при переходе от одного издателя API к другому.

- **settings.apiserver.serviceAccount.additionalAPIIssuers**

массив строк

Список дополнительных издателей API токенов ServiceAccount, которые нужно включить при их создании.

Эти издатели (iss) используются для проверки источника токенов, чтобы гарантировать, что они поступают от доверенных сущностей.

Когда указано несколько издателей, то первый используется для генерации токенов, а все указанные издатели принимаются для валидации токенов.

Этот вариант полезен при переходе от одного API издателя к другому.

- **settings.apiserver.serviceAccount.issuer**

строка

Издатель ServiceAccount'ов. Это URL API-сервера. Значения этого поля используются как `iss` claim токена и для проверки JWT-токенов ServiceAccount.

Издатель по умолчанию автоматически генерируется по шаблону `https://kubernetes.default.svc.${clusterDomain}`, например, `https://kubernetes.default.svc.cluster.local`

Обратите внимание, что все поды в кластере, использующие токены ServiceAccount, должны быть перезапущены при изменении этой опции.

Пример:

```
issuer: https://api.example.com
```

- **settings.enabledFeatureGates**

массив строк

Список включенных в control plane экспериментальных функций (feature gates).

Описание отдельных feature gates доступно в документации Kubernetes. Список поддерживаемых в DKP feature gates доступен на странице описания модуля.

По умолчанию: `[]`

- **Элемент массива**

строка

Шаблон: `^[a-zA-Z]+$`

- **settings.etcd**

объект

Параметры etcd.

- **settings.etcd.backup**

объект

Параметры резервного копирования etcd.

- **settings.etcd.backup.cronSchedule**

строка

Расписание резервного копирования etcd в формате cron. Используется локальный часовой пояс kube-controller-manager.

По умолчанию: `0 0 * * *`

Пример:

```
cronSchedule: 0 1 * * *
```

■ **settings.etcd.backup.enabled**

булевый

Включение резервного копирования etcd.

По умолчанию: `true`

■ **settings.etcd.backup.hostPath**

строка

Путь для хранения резервных копий etcd на master-узлах.

По умолчанию: `/var/lib/etcd`

■ **settings.etcd.maxDbSize**

число

quota-backend-bytes параметр. Deckhouse автоматически управляет quota-backend-bytes параметром. Если параметр maxDbSize установлен, Deckhouse будет использовать это значение для параметра quota-backend-bytes etcd.

Минимальное значение: 512MB.

Максимальное значение: 8GB.

Экспериментальный. Может быть удален в будущем.

Допустимые значения: `536870912 <= X <= 8589934592`

○ **settings.failedNodePodEvictionTimeoutSeconds**

целочисленный

Количество секунд, по прошествии которых с узла в статусе `Unreachable` будут удалены поды.



Важно! При изменении этого параметра потребуется перезапуск подов.

По умолчанию: `300`

○ **settings.nodeMonitorGracePeriodSeconds**

целочисленный

Число секунд, через которое узел перейдет в состояние `Unreachable` при потере с ним связи.

По умолчанию: `40`

○ **settings.rootKubeconfigSymlink**

булевый

Поддерживать ли символическую ссылку `/root/.kube/config` на `/etc/kubernetes/admin.conf` на узлах control-plane.

Когда включён модуль user-authz и этот параметр равен `false`, символическая ссылка удаляется, в подах control-plane-manager выставляется переменная окружения `NODE_ADMIN_KUBECONFIG` соответствующим образом. Для доступа к API с узла используйте `d8 k s -- kubeconfig=/etc/kubernetes/admin.conf` (или другие учётные данные).

Если модуль user-authz выключен, control-plane-manager этот параметр не учитывает и сохраняет поведение по умолчанию (симлинк создаётся).

Переключение обратно на `true` восстанавливает симлинк при следующем цикле согласования.

По умолчанию: `true`

Примеры:

```
rootKubeconfigSymlink: true
```

```
rootKubeconfigSymlink: false
```

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

KubeSchedulerWebhookConfiguration

Scope: Cluster

Version: v1alpha1

Определяет конфигурацию внешнего вебхука планировщика (kube-scheduler).

Подключение внешнего вебхука к планировщику позволяет расширить возможности планировщика, и учитывать более сложные условия при планировании нагрузки в кластере.

- **webhooks**

массив объектов

Обязательный параметр

- **webhooks.clientConfig**

объект

Обязательный параметр

ClientConfig определяет параметры доступа к вебхуку.

- **webhooks.clientConfig.caBundle**

строка

Обязательный параметр

caBundle - это пакет CA в PEM-кодировке, который будет использоваться для проверки сертификата сервера webhook.

Шаблон: `^(?:[A-Za-z0-9+\/]{4})*(?:[A-Za-z0-9+\/]{4}|[A-Za-z0-9+\/]{3}=[A-Za-z0-9+\/]{2}={2})$`

- **webhooks.clientConfig.service**

объект

Обязательный параметр

Параметры доступа к вебхуку через сервис в кластере.

- **webhooks.clientConfig.service.name**

строка

Обязательный параметр

Имя сервиса.

Шаблон: `^[0-9a-z\-]+$`

Длина: 1..255

- **webhooks.clientConfig.service.namespace**

строка

Обязательный параметр

Namespace сервиса.

Шаблон: `^[0-9a-z\-]+$`

Длина: 1..255

- **webhooks.clientConfig.service.path**

строка

Обязательный параметр

URI путь.

Шаблон: `^[0-9a-zA-Z._\/-]+$`

Длина: 1..255

- **webhooks.clientConfig.service.port**

целочисленный

Обязательный параметр

Порт сервиса.

Допустимые значения: `1 <= X <= 65535`

- **webhooks.failurePolicy**

строка

Политика реакции на событие, если вебхук вернул ошибку или недоступен:

- `Fail` — падать с ошибкой;
- `Ignore` — пропускать (игнорировать).

По умолчанию: `Fail`

Допустимые значения: `Fail`, `Ignore`

◦ **`webhooks.filterVerb`**
строка

Verb, который добавляется к `URLPrefix` при выполнении вызова `filter` к расширению планировщика (scheduler extender). Если значение не задано, используется `filter` (для обратной совместимости, без необходимости не менять). Укажите пустую строку `" "`, чтобы отключить вызовы `filter` для этого расширения.

◦ **`webhooks.preemptVerb`**
строка

Verb, который добавляется к `URLPrefix` при выполнении вызова `preempt` к расширению планировщика (scheduler extender). Оставьте пустую строку `" "`, если не нужно выполнять вызовы `preempt` для этого расширения.

◦ **`webhooks.prioritizeVerb`**
строка

Verb, который добавляется к `URLPrefix` при выполнении вызова `prioritize` к расширению планировщика (scheduler extender). Если значение не задано, используется `prioritize` (для обратной совместимости, без необходимости не менять). Укажите пустую строку `" "`, чтобы отключить вызовы `prioritize` для этого расширения.

◦ **`webhooks.timeoutSeconds`**
целочисленный

Таймаут срабатывания вебхука.

По умолчанию: `10`

Допустимые значения: `1 <= X <= 60`

◦ **`webhooks.weight`**
целочисленный

Обязательный параметр

Числовой множитель для оценок узла, которые генерирует вызов расчета приоритизации.

Допустимые значения: `1 <= X`

FAQ

Как добавить master-узел в статическом или гибридном кластере?

i Важно иметь нечетное количество master-узлов для обеспечения кворума.

⚠ Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

В процессе установки Deckhouse Kubernetes Platform с настройками по умолчанию в NodeGroup `master` отсутствует секция `spec.staticInstances.labelSelector` с настройками фильтра лейблов по ресурсам `staticInstances`. Из-за этого после изменения количества узлов `staticInstances` в NodeGroup `master` (параметр `spec.staticInstances.count`) при добавлении обычного узла с помощью Cluster API Provider Static (CAPS) он может быть «перехвачен» и добавлен в NodeGroup `master`, даже если в соответствующем ему StaticInstance (в `metadata`) указан лейбл с `role`, отличающейся от `master`. Чтобы избежать этого «перехвата», после установки DKP измените NodeGroup `master` — добавьте в нее секцию `spec.staticInstances.labelSelector` с настройками фильтра лейблов по ресурсам `staticInstances`. Пример NodeGroup `master` с `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Далее при добавлении в кластер master-узлов с помощью CAPS указывайте в соответствующих им StaticInstance лейбл, заданный в `spec.staticInstances.labelSelector` NodeGroup `master`. Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # Лейбл, указанный в spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Укажите IP-адрес сервера статического узла.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

i При добавлении новых master-узлов с помощью CAPS и изменении в NodeGroup `master` количества master-узлов (параметр `spec.staticInstances.count`) учитывайте следующее:

При первоначальной установке кластера в конфигурации указывается первый master-узел, на который происходит установка. Если после первоначальной установки кластера нужно развернуть несколько master-узлов и добавить их с помощью CAPS, в параметре `spec.staticInstances.count` NodeGroup `master` необходимо указать количество узлов на один меньше желаемого.

Например, если нужен кластер из трёх master-узлов, в `spec.staticInstances.count` NodeGroup `master` укажите значение 2 и создайте два staticInstances для добавляемых узлов. После их добавления в кластер количество master-узлов будет равно трём: master-узел, на который происходила установка и два master-узла, добавленные с помощью CAPS.

В остальном добавление master-узла в статический или гибридный кластер аналогично добавлению обычного узла. Для этого воспользуйтесь соответствующими примерами. Все необходимые действия по настройке компонентов control plane кластера на новых master-узлах выполняются автоматически. Дождитесь появления master-узлов в статусе `Ready`.

Как добавить master-узлы в облачном кластере?

Далее описана конвертация кластера с одним master-узлом в кластер с несколькими master-узлами.

i Перед добавлением узлов убедитесь в наличии необходимых квот.

Важно иметь нечетное количество master-узлов для обеспечения кворума.

⚠ Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать созданию новых master-узлов.

4. Убедитесь, что очередь Deckhouse пуста:

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.



При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика Deckhouse соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду, чтобы проверить состояние перед началом работы:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

Ответ должен сообщить, что Terraform не нашёл расхождений и изменений не требуется.

8. **В контейнере с инсталлятором** выполните следующую команду и укажите требуемое количество master-узлов в параметре `masterNodeGroup.replicas`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST>
```



Для **Yandex Cloud**, при использовании внешних адресов на master-узлах, количество элементов массива в параметре `masterNodeGroup.instanceClass.externalIPAddresses` должно равняться количеству master-узлов. При использовании значения `Auto` (автоматический заказ публичных IP-адресов), количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при трех master-узлах (`masterNodeGroup.replicas: 3`) и автоматическом заказе адресов, параметр `masterNodeGroup.instanceClass.externalIPAddresses` будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
- "Auto"
- "Auto"
```

9. **В контейнере с инсталлятором** выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

10. Дождитесь появления необходимого количества master-узлов в статусе `Ready` и готовности всех экземпляров `control-plane-manager`:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-control-plane-manager
```

Как уменьшить число master-узлов в облачном кластере?

Далее описана конвертация кластера с несколькими master-узлами в кластер с одним master-узлом.



Описанные ниже шаги необходимо выполнять с первого по порядку master-узла кластера (`master-0`). Это связано с тем, что кластер всегда масштабируется по порядку: например, невозможно удалить узлы `master-0` и `master-1`, оставив `master-2`.



Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
4. Убедитесь, что очередь DKP пуста:

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести Username и Password.

i При авторизации в хранилище registry.deckhouse.ru поле Username должно иметь значение license-token, а Password — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду и укажите 1 в параметре masterNodeGroup.replicas:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
--ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

i Для **Yandex Cloud** при использовании внешних адресов на master-узлах количество элементов массива в параметре masterNodeGroup.instanceClass.externalIPAddresses должно равняться количеству master-узлов. При использовании значения Auto (автоматический заказ публичных IP-адресов) количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при одном master-узле (masterNodeGroup.replicas: 1) и автоматическом заказе адресов параметр masterNodeGroup.instanceClass.externalIPAddresses будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
```

8. **В контейнере с инсталлятором** выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

i Для **OpenStack** и **VK Cloud (OpenStack)** после подтверждения удаления узла обязательно проверьте удаление диска <prefix>kubernetes-data-N в самом Openstack.

Например, при удалении узла cloud-demo-master-2 в веб-интерфейсе Openstack или в OpenStack CLI необходимо проверить отсутствие диска cloud-demo-kubernetes-data-2.

В случае, если диск kubernetes-data останется, при увеличении количества master-узлов могут возникнуть проблемы в работе ETCD.

9. Выполните проверку очереди Deckhouse и убедитесь, что отсутствуют ошибки, с помощью команды:

```
d8 system queue list
```

Как убрать роль master-узла, сохранив узел?

⚠ Если в кластере используется модуль stronghold, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

- Сделайте резервную копию etcd и директории /etc/kubernetes.
- Скопируйте полученный архив за пределы кластера (например, на локальную машину).
- Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
- Убедитесь, что очередь Deckhouse пуста:

```
d8 system queue list
```

- Снимите следующие лейблы:
 - node-role.kubernetes.io/control-plane
 - node-role.kubernetes.io/master
 - node.deckhouse.io/group

Команда для снятия лейблов:

```
d8 k label node <MASTER-NODE-N-NAME> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master- node.deckhouse.io/group-
```

- Убедитесь, что удаляемый master-узел пропал из списка членов кластера etcd:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

7. Зайдите на узел и выполните следующие команды:

```
rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
rm -f /etc/kubernetes/authorization-webhook-config.yaml
rm -f /etc/kubernetes/admin.conf /root/.kube/config
rm -rf /etc/kubernetes/deckhouse
rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
rm -rf /var/lib/etcd/member/
```

Как изменить образ ОС в кластере с несколькими master-узлами?

Способ изменения ОС зависит от типа кластера: в облачном кластере узлы заменяются через `dhctl converge`, в статическом — вручную, по одному узлу.

В облачном кластере

Чтобы изменить образ ОС в облачном кластере с несколькими master-узлами, выполните следующие шаги:

 Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
4. Убедитесь, что очередь Deckhouse пуста:

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.

 При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика Deckhouse соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]')
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
  registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду, чтобы проверить состояние перед началом работы:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Ответ должен сообщить, что Terraform не нашёл расхождений и изменений не требуется.

8. **В контейнере с инсталлятором** выполните следующую команду и укажите необходимый образ ОС в параметре `masterNodeGroup.instanceClass` (укажите адреса всех master-узлов в параметре `--ssh-host`):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

9. **В контейнере с инсталлятором** выполните следующую команду, чтобы провести обновление узлов:

Внимательно изучите действия, которые планирует выполнить `converge`, когда запрашивает подтверждение.

При выполнении команды узлы будут заменены на новые с подтверждением на каждом узле. Замена будет выполняться по очереди в обратном порядке (2,1,0).

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
--ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

Следующие действия **выполняйте поочерёдно на каждом** master-узле, начиная с узла с наивысшим номером (с суффиксом 2) и заканчивая узлом с наименьшим номером (с суффиксом 0).

10. На созданном узле откройте журнал systemd-юнита `bashible.service`. Дождитесь окончания настройки узла — в журнале должно появиться сообщение `nothing to do`:

```
journalctl -fu bashible.service
```

11. Проверьте, что узел `etcd` отображился в списке узлов кластера:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

12. Убедитесь, что `control-plane-manager` функционирует на узле.

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
-l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
```

13. Перейдите к обновлению следующего узла.

В статическом кластере

Инструкция ниже описывает замену ОС на master-узлах, которые добавлены в кластер **вручную** с помощью скрипта `bootstrap.sh`. Выполняйте шаги поочерёдно для каждого master-узла и переходите к следующему только после того, как текущий узел вернулся в кластер и стал работоспособен.

⚠ Если master-узлы управляются Cluster API Provider Static (CAPS) через ресурсы `StaticInstance`, не используйте эту инструкцию. Сначала удалите `StaticInstance`, установите требуемую ОС и снова добавьте узел в `NodeGroup` `master`.

⚠ Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений настоятельно рекомендуется создать резервную копию данных модуля.

Чтобы изменить ОС вручную добавленного master-узла, выполните следующие шаги:

1. Сделайте резервную копию `etcd` и директории `/etc/kubernetes`. Если используется модуль `stronghold`, убедитесь, что создана резервная копия его данных.
2. Проверьте состояние кластера, отсутствие алертов и незавершённых задач в очереди Deckhouse:

```
d8 status
```

3. Снимите с узла лейблы master-узла:

```
d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
node-role.kubernetes.io/master- node.deckhouse.io/group-
```

где `<MASTER_NODE_NAME>` — имя изменяемого master-узла.

4. Убедитесь, что узел удалён из списка членов кластера `etcd`:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

5. Вытесните нагрузку с узла командой `d8 k drain`:

```
d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
```

6. Принудительно удалите оставшиеся на узле поды:

```
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --force
```

7. Удалите объект `Node`:

```
d8 k delete node <MASTER_NODE_NAME>
```

- На удаляемом master-узле очистите данные DKP. Команда необратимо удаляет данные Kubernetes и DKP с узла. Перед выполнением убедитесь, что выбран правильный узел и созданы необходимые резервные копии:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

- Установите на узле требуемую ОС.
- На одном из оставшихся в кластере master-узлов получите и раскодируйте скрипт для добавления master-узла:

```
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
-o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
```

- Безопасно скопируйте полученный на предыдущем шаге файл `bootstrap.sh` на добавляемый узел и выполните его на этом узле от пользователя `root`:

```
bash bootstrap.sh
```

- На добавляемом узле** откройте журнал systemd-юнита `bashible.service`. Дождитесь окончания настройки узла — в журнале должно появиться сообщение `nothing to do`:

```
journalctl -fu bashible.service
```

- Дождитесь перехода узла в статус `Ready`:

```
d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
```

- Проверьте, что узел отобразился в списке членов кластера etcd:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
    --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

- Убедитесь, что `control-plane-manager` функционирует на узле:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
-l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
```

- Убедитесь, что в кластере нет алертов и незавершённых задач:

```
d8 status
```

- Повторите процедуру для следующего master-узла.

Как изменить образ ОС в кластере с одним master-узлом?

Способ зависит от типа кластера: сначала добавьте дополнительные master-узлы, замените ОС в мультимастерном режиме, затем верните исходное число master-узлов.



Если в кластере используется модуль `stronghold`, перед изменением master-узлов убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом изменений настоятельно рекомендуется создать резервную копию данных модуля.

В облачном кластере

- Преобразуйте кластер с одним master-узлом в кластер с несколькими master-узлами в соответствии с инструкцией.
- Измените ОС на master-узлах в соответствии с инструкцией.
- Преобразуйте кластер с несколькими master-узлами в кластер с одним master-узлом в соответствии с инструкцией.

В статическом кластере

- Добавьте дополнительные master-узлы в соответствии с инструкцией.
- Измените ОС на master-узлах в соответствии с инструкцией.
- Выведите лишние master-узлы из роли control plane в соответствии с инструкцией. Затем удалите их из кластера командой `d8 k delete node <MASTER_NODE_NAME>` и выключите соответствующие серверы.

Как настроить режим HA с двумя master-узлами и arbiter-узлом?

В Deckhouse Kubernetes Platform возможна настройка режима HA с двумя master-узлами и arbiter-узлом. Такой подход позволяет обеспечить требования по HA в условиях ограниченных ресурсов.

На arbiter-узле размещается только etcd, без остальных компонентов control plane. Этот узел используется для обеспечения кворума etcd.

Требования к arbiter-узлу:

- не менее 2 ядер CPU;
- не менее 4 ГБ RAM;
- не менее 8 ГБ дискового пространства под etcd.

Требования к сетевым задержкам для arbiter-узла аналогичны требованиям для master-узлов.

Настройка в облачном кластере

Пример ниже актуален для облачного кластера с тремя master-узлами. Чтобы настроить режим HA с двумя master-узлами и arbiter-узлом в облачном кластере, необходимо удалить из кластера один master-узел и добавить один arbiter-узел.

⚠ Описанные ниже шаги необходимо выполнять с первого по порядку master-узла кластера (`master-0`). Это связано с тем, что кластер всегда масштабируется по порядку: например, невозможно удалить узлы `master-0` и `master-1`, оставив `master-2`.

⚠ Если в кластере используется модуль `stronghold`, перед добавлением или удалением master-узла убедитесь, что модуль находится в полностью работоспособном состоянии. Перед началом любых изменений рекомендуется создать резервную копию данных модуля.

1. Сделайте резервную копию etcd и директории `/etc/kubernetes`.
2. Скопируйте полученный архив за пределы кластера (например, на локальную машину).
3. Убедитесь, что в кластере нет алертов, которые могут помешать обновлению master-узлов.
4. Убедитесь, что очередь DKP пуста:

```
d8 system queue list
```

5. **На локальной машине** авторизуйтесь в хранилище образов контейнеров (измените адрес хранилища образов при необходимости):

```
docker login registry.deckhouse.ru
```

В процессе авторизации необходимо будет ввести `Username` и `Password`.

i При авторизации в хранилище `registry.deckhouse.ru` поле `Username` должно иметь значение `license-token`, а `Password` — содержать ключ лицензии Deckhouse Kubernetes Platform.

6. **На локальной машине** запустите контейнер установщика DKP соответствующей редакции и версии (измените адрес хранилища образов при необходимости):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core.deckhouse.io/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core.deckhouse.io/edition}' | tr '[:upper:]' '[:lower:]')
docker run --pull=always -it -v "$HOME/.ssh:/tmp/.ssh/" \
registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **В контейнере с инсталлятором** выполните следующую команду:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
--ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

Измените настройки облачного провайдера:

- В параметре `masterNodeGroup.replicas` укажите `2`.
- Создайте NodeGroup для arbiter-узла. На arbiter-узле **обязательно** должен быть лейбл `node.deckhouse.io/etcd-arbiter: ""` и taint, предотвращающий размещение на нем пользовательской нагрузки. Пример описания NodeGroup для arbiter-узла:

```
nodeGroups:
- name: arbiter
  replicas: 1
  nodeTemplate:
    labels:
      node.deckhouse.io/etcd-arbiter: ""
    taints:
      - key: node.deckhouse.io/etcd-arbiter
        effect: NoSchedule
  zones:
    - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
# ... остальная часть манифеста
```

- Сохраните изменения.

i Для **Yandex Cloud** при использовании внешних адресов на master-узлах количество элементов массива в параметре `masterNodeGroup.instanceClass.externalIPAddresses` должно равняться количеству master-узлов. При использовании значения `Auto` (автоматический заказ публичных IP-адресов) количество элементов в массиве все равно должно соответствовать количеству master-узлов.

Например, при одном master-узле (`masterNodeGroup.replicas: 1`) и автоматическом заказе адресов параметр `masterNodeGroup.instanceClass.externalIPAddresses` будет выглядеть следующим образом:

```
externalIPAddresses:
- "Auto"
```

8. В контейнере с инсталлятором выполните следующую команду для запуска масштабирования:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
--ssh-host <MASTER-NODE-1-HOST>
```

i **Важно.** Для **OpenStack** и **VK Cloud (OpenStack)** после подтверждения удаления узла обязательно проверьте удаление диска `<prefix>kubernetes-data-N` в самом OpenStack.

Например, при удалении узла `cloud-demo-master-2` в веб-интерфейсе OpenStack или в OpenStack CLI необходимо проверить отсутствие диска `cloud-demo-kubernetes-data-2`.

В случае, если диск `kubernetes-data` останется, при увеличении количества master-узлов могут возникнуть проблемы в работе etcd.

9. Проверьте очередь Deckhouse с помощью следующей команды и убедитесь, что отсутствуют ошибки:

```
d8 system queue list
```

Настройка в статическом кластере

Чтобы настроить режим HA с двумя master-узлами и arbiter-узлом в статическом кластере, выполните следующие действия:

1. Создайте NodeGroup для arbiter-узла. На arbiter-узле **обязательно** должен быть лейбл `node.deckhouse.io/etcd-arbiter: ""` и taint, предотвращающий размещение на нем пользовательской нагрузки. Пример описания NodeGroup для arbiter-узла:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: arbiter
spec:
  nodeType: Static
  nodeTemplate:
    labels:
      node.deckhouse.io/etcd-arbiter: ""
    taints:
      - key: node.deckhouse.io/etcd-arbiter
        effect: NoSchedule
# ... остальная часть манифеста
```

2. Добавьте удобным вам способом в кластер узел, который будет использовать как arbiter-узел.
3. Убедитесь, что добавленный arbiter-узел находится в списке членов кластера etcd.
4. Удалите один master-узел из кластера.

Как посмотреть список узлов кластера в etcd?

Вариант 1

Используйте команду `etcdctl member list`.

Пример:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -W table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Внимание. Последний параметр в таблице вывода показывает, что узел находится в состоянии `learner`, а не в состоянии `leader`.

Вариант 2

Используйте команду `etcdctl endpoint status`. Для лидера в столбце `IS LEADER` будет указано значение `true`.

Пример:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Что делать, если что-то пошло не так?

В процессе работы `control-plane-manager` автоматически создает резервные копии конфигурации и данных, которые могут пригодиться в случае возникновения проблем. Эти резервные копии сохраняются в директории `/etc/kubernetes/deckhouse/backup`. Если в процессе работы возникли ошибки или непредвиденные ситуации, вы можете использовать эти резервные копии для восстановления до предыдущего исправного состояния.

Что делать, если кластер etcd не функционирует?

Если кластер `etcd` не функционирует и не удается восстановить его из резервной копии, вы можете попытаться восстановить его с нуля, следуя шагам ниже.

1. Сначала на всех узлах, которые являются частью вашего кластера `etcd`, кроме одного, удалите манифест `etcd.yaml`, который находится в директории `/etc/kubernetes/manifests/`. После этого только один узел останется активным, и с него будет происходить восстановление состояния кластера с несколькими master-узлами.
2. На оставшемся узле откройте файл манифеста `etcd.yaml` и укажите параметр `--force-new-cluster` в `spec.containers.command`.
3. После успешного восстановления кластера, удалите параметр `--force-new-cluster`.



Эта операция является деструктивной, так как она полностью уничтожает текущие данные и инициализирует кластер с состоянием, которое сохранено на узле. Все pending-записи будут утеряны.

Что делать, если etcd постоянно перезапускается с ошибкой?

Этот способ может понадобиться, если использование параметра `--force-new-cluster` не восстанавливает работу `etcd`. Это может произойти, если `converge` master-узлов прошел неудачно, в результате чего новый master-узел был создан на старом диске `etcd`, изменил свой адрес в локальной сети, а другие master-узлы отсутствуют. Этот метод стоит использовать если контейнер `etcd` находится в бесконечном цикле перезапуска, а в его логах появляется ошибка: `panic: unexpected removal of unknown remote peer`.

1. Найдите утилиту `etcdctl` на master-узле и скопируйте исполняемый файл в `/usr/local/bin/`:

```
cp $(find /var/lib/containerd/ \
-name etcdctl -print -quit) /usr/local/bin/etcdctl
```

2. Создайте новый снимок базы `etcd` на основе текущего локального снимка (`/var/lib/etcd/member/snap/db`):

```
etcdctl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
--initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-urls=https://<ADDRESS>:2380 \
--skip-hash-check=true --data-dir /var/lib/etcdtest
```

где:

- `<HOSTNAME>` — имя master-узла;
- `<ADDRESS>` — адрес master-узла.

3. Выполните следующие команды для использования нового снимка:

```
cp -r /var/lib/etcd /tmp/etcd-backup
rm -rf /var/lib/etcd
mv /var/lib/etcdtest /var/lib/etcd
```

4. Найдите контейнеры `etcd` и `api-server`:

```
crictl ps -a | egrep "etcd|apiserver"
```

5. Удалите найденные контейнеры `etcd` и `api-server`:

```
crictl rm <CONTAINER-ID>
```

6. Перезапустите master-узел.

Что делать, если объем базы данных etcd достиг лимита, установленного в quota-backend-bytes?

Когда объем базы данных `etcd` достигает лимита, установленного параметром `quota-backend-bytes`, доступ к ней становится "read-only". Это означает, что база данных `etcd` перестает принимать новые записи, но при этом остается доступной для чтения данных. Вы можете понять, что столкнулись с подобной ситуацией, выполнив команду:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
  etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ \
  endpoint status -w table --cluster
```

Если в поле `ERRORS` вы видите подобное сообщение `alarm:NOSPACE`, значит вам нужно предпринять следующие шаги:

1. На **каждом master-узле** (из полученного списка членов кластера etcd) найдите строку с `--quota-backend-bytes` в файле манифеста пода etcd по пути `/etc/kubernetes/manifests/etcd.yaml` и увеличьте значение, умножив указанный параметр в этой строке на два. Если такой строки нет — добавьте, например: `--quota-backend-bytes=8589934592`. Эта настройка задает лимит на 8 ГБ.
2. Сбросьте активное предупреждение (alarm) о нехватке места в базе данных. Для этого выполните следующую команду:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p) -- \
  etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ alarm disarm
```

3. Измените параметр `maxDbSize` в настройках `control-plane-manager` на тот, который был задан в манифесте.

Как сжать базу данных etcd



Перед операцией сжатия базы данных etcd создайте резервную копию etcd.

Для просмотра размера БД etcd на определенном узле до операции и после неё используйте команду (здесь `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  endpoint status --cluster -w table
```

Пример вывода (размер БД etcd на узле указывается в колонке `DB SIZE`):

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          ENDPOINT          |      ID      | VERSION | STORAGE VERSION | DB SIZE | IN USE | PERCENTAGE NOT | | |
| IN USE | QUOTA | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS | DOWNGRADE |
| TARGET VERSION | DOWNGRADE ENABLED |          |          |          |          |          |          |          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB | 20% | 2.1 GB | true | false | 56 | 258054684 | 258054684 |  |
|          false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB | 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |  |
|          false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB | 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |  |
|          false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Как сжать базу данных etcd на узле в кластере с одним master-узлом



Сжатие базы данных etcd — операция с высокой нагрузкой на ресурсы узла, которая на время полностью блокирует работу etcd на данном узле. Учитывайте это при выборе времени для проведения операции в кластере с одним master-узлом.

Чтобы выполнить сжатие базы данных etcd в кластере с одним master-узлом, используйте следующую команду (здесь `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Пример вывода при успешном выполнении операции:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

i При появлении ошибки из-за таймаута увеличивайте значение параметра `--command-timeout` из команды выше, пока операция не завершится успешно.

Как сжать базу данных etcd в кластере с несколькими master-узлами

Чтобы выполнить сжатие базы данных etcd в кластере с несколькими master-узлами:

1. Получите список подов etcd. Для этого используйте следующую команду:

```
d8 k -n kube-system get pod -l component=etcd -o wide
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS
GATES								
etcd-master-0	1/1	Running	0	3d21h	192.168.199.80	master-0	<none>	<none>
etcd-master-1	1/1	Running	0	3d21h	192.168.199.81	master-1	<none>	<none>
etcd-master-2	1/1	Running	0	3d21h	192.168.199.82	master-2	<none>	<none>

2. Определите master-узел — лидер. Для этого обратитесь к любому поду etcd и получите список узлов — участников кластера etcd с помощью команды (где `NODE_NAME` — имя master-узла):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
endpoint status --cluster -w table
```

Пример вывода (у лидера в колонке `IS LEADER` будет значение `true`):

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          |          |          |          |          |          |          |          |          |
| ENDPOINT | ID       | VERSION | STORAGE | DB SIZE | IN USE | PERCENTAGE |
| NOT IN USE | QUOTA | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
| DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | true | false | 56 | 258054684 | 258054684 |
| false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |
| false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 | 3.6.1 | 3.6.0 | 76 MB | 62 MB |
| 20% | 2.1 GB | false | false | 56 | 258054685 | 258054685 |
| false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

3. Поочередно выполните сжатие базы данных etcd на узлах — участниках кластера etcd. Используйте команду (здесь `NODE_NAME` — имя master-узла):

i Важно: операцию на узле-лидере выполняйте в последнюю очередь.
Восстановление etcd на узле после операции может занять некоторое время. Рекомендуется подождать не менее минуты прежде чем переходить к следующему узлу.

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt \
--key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Пример вывода при успешном выполнении операции:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

i При появлении ошибки из-за таймаута увеличивайте значение параметра `--command-timeout` из команды выше, пока операция не завершится успешно.

Модель административного доступа к кластеру

Модуль `control-plane-manager` поддерживает несколько файлов `kubeconfig` на master-узлах. Понимание их назначения важно для безопасного администрирования кластера.

Файлы `kubeconfig` на master-узлах

Файл	Идентификация	Назначение
<code>/etc/kubernetes/admin.conf</code>	<code>kubernetes-admin</code> (группа <code>kubeadm:cluster-admins</code>)	Машинный <code>kubeconfig</code> для внутренних операций <code>kubeadm</code> (<code>join</code> , обновление). Группа привязана к встроенной wildcard-роли <code>cluster-admin</code> .
<code>/etc/kubernetes/super-admin.conf</code>	<code>kubernetes-super-admin</code> (группа <code>system:masters</code>)	Аварийный доступ (<code>break-glass</code>). Обходит RBAC полностью. Ограничьте доступ к файлу сценариями восстановления.
<code>/etc/kubernetes/controller-manager.conf</code>	<code>system:kube-controller-manager</code>	Используется <code>kube-controller-manager</code> .
<code>/etc/kubernetes/scheduler.conf</code>	<code>system:kube-scheduler</code>	Используется <code>kube-scheduler</code> .

Административный доступ на основе RBAC

Начиная с Kubernetes 1.29, `kubeadm` генерирует `admin.conf` с группой `kubeadm:cluster-admins` вместо `system:masters`. Это обеспечивает управляемый через RBAC административный доступ, который может быть отозван путём удаления объектов привязки RBAC для `kubeadm:cluster-admins` (или нескольких таких записей).

Deckhouse привязывает группу `kubeadm:cluster-admins` к встроенной wildcard-роли `cluster-admin` (как в обычном кластере `kubeadm` без дополнительной настройки RBAC) независимо от того, включён ли модуль `user-authz`. Для полного неограниченного доступа в обход RBAC используйте `super-admin.conf`.

Рекомендуемый административный доступ

Когда модуль `user-authn` включён, используйте персонализированный `kubeconfig` на основе OIDC, получаемый через `kubeconfig`-генератор. Это обеспечивает индивидуальную ответственность и журнал аудита.

Когда `user-authn` отключён, администраторы могут явно использовать `admin kubeconfig` на master-узле:

```
d8 k --kubeconfig=/etc/kubernetes/admin.conf <команда>
```

Символическая ссылка `root kubeconfig`

По умолчанию модуль CPM создаёт символическую ссылку `/root/.kube/config -> /etc/kubernetes/admin.conf` на master-узлах, что позволяет root-пользователю запускать `d8 k` без указания `--kubeconfig`.

Когда модуль `user-authz` включён, это поведение можно отключить, задав `rootKubeconfigSymlink: false` в конфигурации модуля **control-plane-manager**:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

Если модуль `user-authz` выключен, CPM этот параметр не использует и сохраняет поведение по умолчанию (символическая ссылка создаётся).

При отключении символической ссылки (при включённом `user-authz`) ссылка удаляется, если она указывала на `admin.conf`. Используйте персонализированные учётные данные или явно указывайте `--kubeconfig`.

Усиление безопасности

Модуль CPM автоматически ограничивает права доступа к файлам `admin.conf` и `super-admin.conf` до `0600` (чтение/запись только для владельца) при каждом цикле согласования. Это предотвращает несанкционированный доступ к этим

конфиденциальным учётным данным.

Аварийный доступ (break-glass)

В экстренных ситуациях (ошибки конфигурации RBAC, отказ webhook'ов) используйте `super-admin.conf`:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <команда>
```

Эти учётные данные обходят все проверки RBAC. Используйте их только в крайнем случае и ограничьте круг лиц с доступом к файлу.

Как настроить дополнительные политики аудита?

1. Включите параметр `auditPolicyEnabled` в настройках модуля:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    apiserver:
      auditPolicyEnabled: true
```

2. Создайте Secret `kube-system/audit-policy` с YAML-файлом политик, закодированным в Base64:

```
apiVersion: v1
kind: Secret
metadata:
  name: audit-policy
  namespace: kube-system
data:
  audit-policy.yaml: <base64>
```

Минимальный рабочий пример `audit-policy.yaml` выглядит так:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  omitStages:
  - RequestReceived
```

С подробной информацией по настройке содержимого файла `audit-policy.yaml` можно ознакомиться:

- В официальной документации Kubernetes;
- В статье на Habr;
- В коде скрипта-генератора, используемого в GCE.

Как исключить встроенные политики аудита?

Установите параметр `apiserver.basicAuditPolicyEnabled` модуля в `false`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

Как вывести аудит-лог в стандартный вывод вместо файлов?

Установите параметр `apiserver.auditLog.output` модуля в значение `Stdout`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

Как работать с журналом аудита?

Предполагается, что на master-узлах установлен «скрейпер логов»: `log-shipper`, `promtail`, `filebeat`, который будет мониторить файл с логами:

```
/var/log/kube-audit/audit.log
```

Параметры ротации логов в файле журнала предустановлены и их изменение не предусмотрено:

- Максимальное занимаемое место на диске `1000` МБ.
- Максимальная глубина записи `30` дней.

В зависимости от настроек политики (`Policy`) и количества запросов к `apiserver` логов может быть очень много, соответственно глубина хранения может быть менее 30 минут.

⚠ Текущая реализация функционала не гарантирует безопасность, так как существует риск временного нарушения работы `control plane`.

Если в `Secret`'е с конфигурационным файлом окажутся неподдерживаемые опции или опечатка, `apiserver` не сможет запуститься.

В случае возникновения проблем с запуском `apiserver`, потребуется вручную отключить параметры `--audit-log-*` в манифесте `/etc/kubernetes/manifests/kube-apiserver.yaml` и перезапустить `apiserver` следующей командой:

```
docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')
# Или (в зависимости используемого вами CRI).
crictl stopp $(crictl pods --name=kube-apiserver -q)
```

После перезапуска будет достаточно времени исправить `Secret` или удалить его:

```
d8 k -n kube-system delete secret audit-policy
```

Как ускорить перезапуск подов при потере связи с узлом?

По умолчанию, если узел в течении 40 секунд не сообщает свое состояние, он помечается как недоступный. И еще через 5 минут поды узла начнут перезапускаться на других узлах. В итоге общее время недоступности приложений составляет около 6 минут.

В специфических случаях, когда приложение не может быть запущено в нескольких экземплярах, есть способ сократить период их недоступности:

1. Уменьшить время перехода узла в состояние `Unreachable` при потере с ним связи настройкой параметра `nodeMonitorGracePeriodSeconds`.
2. Установить меньший таймаут удаления подов с недоступного узла в параметре `failedNodePodEvictionTimeoutSeconds`.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```

В этом случае при потере связи с узлом приложения будут перезапущены примерно через 1 минуту.

Оба упомянутых параметра напрямую влияют на использование процессора и памяти `control-plane`'ом. Снижая таймауты, системные компоненты чаще отправляют статусы и проверяют состояние ресурсов.

При выборе оптимальных значений учитывайте графики использования ресурсов управляющих узлов. Чем меньше значения параметров, тем больше ресурсов может понадобиться для их обработки на этих узлах.

Резервное копирование и восстановление etcd

Что выполняется автоматически

Автоматически запускаются `CronJob` `kube-system/d8-etcd-backup-*` в 00:00 по UTC+0. Результат сохраняется в `/var/lib/etcd/etcd-backup.tar.gz` на всех узлах с `control-plane` в кластере (master-узлы).

Как сделать резервную копию etcd вручную**Используя Deckhouse CLI (Deckhouse Kubernetes Platform v1.65+)**

Начиная с релиза Deckhouse Kubernetes Platform v1.65, стала доступна утилита `d8 backup etcd`, которая предназначена для быстрого создания снимков состояния etcd.

```
d8 backup etcd ./etcd-backup.snapshot
```

Используя bash (Deckhouse Kubernetes Platform v1.64 и старше)

Войдите на любой control-plane узел под пользователем `root` и используйте следующий bash-скрипт:

```
#!/usr/bin/env bash
set -e

pod=etcd-`hostname`
d8 k -n kube-system exec "$pod" -- /usr/bin/etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt -
-key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ snapshot save /var/lib/etcd/${pod##*/}.snapshot && \
mv /var/lib/etcd/"${pod##*/}.snapshot" etcd-backup.snapshot && \
cp -r /etc/kubernetes/ ./ && \
tar -cvzf kube-backup.tar.gz ./etcd-backup.snapshot ./kubernetes/
rm -r ./kubernetes ./etcd-backup.snapshot
```

В текущей директории будет создан файл `kube-backup.tar.gz` со снимком базы etcd одного из узлов кластера. Из полученного снимка можно будет восстановить состояние кластера.

Рекомендуем сделать резервную копию директории `/etc/kubernetes`, в которой находятся:

- манифесты и конфигурация компонентов control-plane;
- PKI кластера Kubernetes.

Данная директория поможет быстро восстановить кластер при полной потере control-plane узлов без создания нового кластера и без повторного присоединения узлов в новый кластер.

Рекомендуем хранить резервные копии снимков состояния кластера etcd, а также резервную копию директории `/etc/kubernetes/` в зашифрованном виде вне кластера Deckhouse. Для этого вы можете использовать сторонние инструменты резервного копирования файлов, например Restic, Borg, Duplicity и т.д.

О возможных вариантах восстановления состояния кластера из снимка etcd вы можете узнать в документации.

Как выполнить полное восстановление состояния кластера из резервной копии etcd?

Далее описаны шаги по восстановлению кластера до предыдущего состояния из резервной копии при полной потере данных.

Восстановление кластера с одним master-узлом

Для корректного восстановления выполните следующие шаги на master-узле:

1. Найдите утилиту `etcdctl` на master-узле и скопируйте исполняемый файл в `/usr/local/bin/`:

```
cp $(find /var/lib/containerd/ \
-name etcdctl -print -quit) /usr/local/bin/etcdctl
```

Проверьте версию `etcdctl` с помощью команды:

```
etcdctl version
```

Должен отобразиться корректный вывод `etcdctl version` без ошибок.

Также вы можете загрузить исполняемый файл `etcdctl` на сервер (желательно, чтобы версия `etcdctl` была такая же, как и версия etcd в кластере):

```
wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
tar -xvzf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdctl /usr/local/bin/etcdctl
```

Проверить версию etcd в кластере можно выполнив следующую команду (команда может не сработать, если etcd и Kubernetes API недоступны):

```
d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdctl version
```

2. Остановите etcd.

```
mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
```

3. Сохраните текущие данные etcd.

```
cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
```

4. Скопируйте или перенесите файл резервной копии etcd `etcd-backup.snapshot` в домашнюю директорию текущего пользователя (`root`):


```
cp /путь/до/резервной/копии/etcd-backup.snapshot ~/etcd-backup.snapshot
```

- Очистите директорию etcd.

```
rm -rf /var/lib/etcd
```

- Восстановите базу данных etcd.

```
ETCDCTL_API=3 etcdctl snapshot restore ~/etcd-backup.snapshot --data-dir=/var/lib/etcd
```

- Запустите etcd. Запуск может занять некоторое время.

```
mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
```

Чтобы убедиться, что etcd запущена, воспользуйтесь командой:

```
crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
```

Пример вывода:

CONTAINER	IMAGE	CREATED	STATE	NAME	ATTEMPT	POD ID	POD
4b11d6ea0338f-gs-test	16d0a07aa1e26	About a minute ago	Running	etcd	0	ee3c8c7d7bba6	etcd-

- Перезапустите master-узел.

Восстановление кластера с несколькими master-узлами

Для корректного восстановления кластера с несколькими master-узлами выполните следующие шаги:

- Активируйте режим High Availability (HA). Это необходимо, чтобы сохранить хотя бы одну реплику Prometheus и его PVC, поскольку в кластере с одним master-узлом HA по умолчанию отключён.
- Переведите кластер в режим с одним master-узлом:
 - В облачном кластере воспользуйтесь инструкцией.
 - В статическом кластере выведите лишние master-узлы из роли `control-plane` по инструкции, после чего удалите их из кластера.
 - В статическом кластере с настроенным режимом HA на базе двух master-узлов и arbiter-узла удалите arbiter-узел и лишние master-узлы.
 - В облачном кластере с настроенным режимом HA на базе двух master-узлов и arbiter-узла воспользуйтесь инструкцией для удаления лишних master-узлов и arbiter-узла.
- Восстановите etcd из резервной копии на единственном оставшемся master-узле. Следуйте инструкции для кластера с одним master-узлом.
- Когда работа etcd будет восстановлена, удалите из кластера информацию об уже удаленных в первом пункте master-узлах, воспользовавшись следующей командой (укажите название узла):

```
d8 k delete node <ИМЯ_MASTER_УЗЛА>
```

Внимание. При недоступности команд `d8 k` или `kubectl` на узле проверьте конфиг `/etc/kubernetes/kubernetes-api-проху/nginx.conf`, в нем должен быть указан только ваш текущий API-сервер. Если в конфиге содержатся IP-адреса старых мастеров, то удалите строчки содержащие их, в этом случае конфиг так же нужно исправить на всех остальных узлах.

- Перезапустите master-узел. Убедитесь, что остальные узлы перешли в состояние `Ready`.
- Дождитесь выполнения заданий из очереди Deckhouse:

```
d8 system queue main
```

- Переведите кластер обратно в режим с несколькими master-узлами. Для облачных кластеров используйте инструкцию. Для статических кластеров используйте инструкцию.

После этих шагов кластер будет успешно восстановлен в конфигурации с несколькими master-узлами.

Как восстановить объект Kubernetes из резервной копии etcd?

Чтобы получить данные определенных объектов кластера из резервной копии etcd:

- Запустите временный экземпляр etcd.
- Заполните его данными из резервной копии.
- Получите описания нужных объектов с помощью `auger`.

Пример шагов по восстановлению объектов из резервной копии etcd

В следующем примере `etcd-backup.snapshot` — резервная копия etcd (snapshot), `infra-production` — пространство имен, в котором нужно восстановить объекты.

- Для загрузки бинарных данных из etcd потребуется утилита `auger`. Ее можно собрать из исходного кода на любой машине с Docker (на узлах кластера это сделать невозможно).

```
git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
cd auger
make release
build/auger -h
```

- Получившийся исполняемый файл `build/auger`, а также `snapshot` из резервной копии etcd нужно загрузить на master-узел, с которого будет выполняться дальнейшие действия.

Данные действия выполняются на master-узле в кластере, на который предварительно был загружен файл `snapshot` и утилита `auger`:

1. Установите корректные права доступа для файла с резервной копией:

```
chmod 644 etcd-backup.snapshot
```

2. Установите полный путь до `snapshot` и до утилиты в переменных окружения:

```
SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot
AUGER_BIN=/root/auger
chmod +x $AUGER_BIN
```

3. Запустите под с временным экземпляром etcd:

- Создайте манифест пода. Он будет запускаться именно на текущем master-узле, выбрав его по переменной `$HOSTNAME`, и смонтирует `snapshot` по пути `$SNAPSHOT` для загрузки во временный экземпляр etcd:

```
cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
    - operator: Exists
  initContainers:
    - command:
      - etcdctl
      - snapshot
      - restore
      - "/tmp/etcd-snapshot"
      - "--data-dir=/default.etcd"
      image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
      imagePullPolicy: IfNotPresent
      name: etcd-snapshot-restore
      # Раскомментируйте фрагмент ниже, чтобы задать лимиты для контейнера, если ресурсов узла
      # недостаточно для его запуска.
      # resources:
      #   requests:
      #     ephemeral-storage: "200Mi"
      #   limits:
      #     ephemeral-storage: "500Mi"
      volumeMounts:
        - name: etcd-dir
          mountPath: /default.etcd
        - name: etcd-snapshot
          mountPath: /tmp/etcd-snapshot
          readOnly: true
    containers:
      - command:
        - etcd
        image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
        imagePullPolicy: IfNotPresent
        name: etcd-temp
        volumeMounts:
          - name: etcd-dir
            mountPath: /default.etcd
      volumes:
        - name: etcd-dir
          emptyDir: {}
        # Используйте фрагмент ниже вместо emptyDir: {}, чтобы задать лимиты для контейнера, если ресурсов
        # узла недостаточно для его запуска.
        # emptyDir:
        #   sizeLimit: 500Mi
      - name: etcd-snapshot
        hostPath:
          path: $SNAPSHOT
          type: File
EOF
```

- Запустите под:

```
d8 k create -f etcd.pod.yaml
```

4. Установите нужные переменные. В текущем примере:

- `infra-production` - пространство имен, в котором мы будем искать ресурсы.
- `/root/etcd-restore/output` - каталог для восстановленных манифестов.
- `/root/auger` - путь до исполняемого файла утилиты `auger`:

```
FILTER=infra-production
BACKUP_OUTPUT_DIR=/root/etcd-restore/output
mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
```

5. Следующие команды отфильтруют список нужных ресурсов по переменной `$FILTER` и выгрузят их в каталог `$BACKUP_OUTPUT_DIR`:

```
files=$(d8 k -n default exec etcdrestore -C etcd-temp -- etcdctl --endpoints=localhost:2379 get / --prefix --keys-only |
grep "$FILTER")
for file in "${files[@]}"
do
    OBJECT=$(d8 k -n default exec etcdrestore -C etcd-temp -- etcdctl --endpoints=localhost:2379 get "$file" --print-
value-only | $AUGER_BIN decode)
    FILENAME=$(echo $file | sed -e "s#/registry/###g;s#/#_#g")
    echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
    echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
done
```

6. Удалите из полученных описаний объектов информацию о времени создания (`creationTimestamp`), `UID`, `status` и прочие оперативные данные, после чего восстановите объекты:

```
d8 k create -f deployments_infra-production_supercronic.yaml
```

7. Удалите под с временным экземпляром etcd:

```
d8 k -n default delete pod etcdrestore
```

Как выбирается узел, на котором будет запущен под?

За распределение подов по узлам отвечает планировщик Kubernetes (компонент `scheduler`). Он проходит через две основные фазы — `Filtering` и `Scoring` (на самом деле, фаз больше, например, `pre-filtering` и `post-filtering`, но в общем можно выделить две ключевые фазы).

Общее устройство планировщика Kubernetes

Планировщик состоит из плагинов, которые работают в рамках какой-либо фазы (фаз).

Примеры плагинов:

- **ImageLocality** — отдаёт предпочтение узлам, на которых уже есть образы контейнеров, которые используются в запускаемом поде. Фаза: `Scoring`.
- **TaintToleration** — реализует механизм taints and tolerations. Фазы: `Filtering`, `Scoring`.
- **NodePorts** — проверяет, есть ли у узла свободные порты, необходимые для запуска пода. Фаза: `Filtering`.

С полным списком плагинов можно ознакомиться в документации Kubernetes.

Логика работы

Профили планировщика

Есть два преднастроенных профиля планировщика:

- `default-scheduler` — профиль по умолчанию, который распределяет поды на узлы с наименьшей загрузкой;
- `high-node-utilization` — профиль, при котором поды размещаются на узлах с наибольшей загрузкой.

Чтобы задать профиль планировщика, укажите его параметре `spec.schedulerName` манифеста пода.

Пример использования профиля:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
    - name: example-pod
      image: registry.k8s.io/pause:2.0
```

Этапы планирования подов

На первой фазе — `Filtering` — активируются плагины фильтрации (filter-плагины), которые из всех доступных узлов выбирают те, которые удовлетворяют определенным условиям фильтрации (например, `taints`, `nodePorts`, `nodeName`, `unschedulable` и другие). Если узлы расположены в разных зонах, планировщик чередует выбор зон, чтобы избежать размещения всех подов в одной зоне.

Предположим, что узлы распределяются по зонам следующим образом:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

В этом случае они будут выбираться в следующем порядке:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4
```

Обратите внимание, что с целью оптимизации выбираются не все попадающие под условия узлы, а только их часть. По умолчанию функция выбора количества узлов линейная. Для кластера из ≤50 узлов будут выбраны 100% узлов, для кластера из 100 узлов — 50%, а для кластера из 5000 узлов — 10%. Минимальное значение — 5% при количестве узлов более 5000. Таким образом, при настройках по умолчанию узел может не попасть в список возможных узлов для запуска.

Эту логику можно изменить (см. подробнее про параметр `percentageOfNodesToScore` в документации Kubernetes), но Deckhouse не дает такой возможности.

После того как были выбраны узлы, соответствующие условиям фильтрации, запускается фаза `Scoring`. Каждый плагин анализирует список отфильтрованных узлов и назначает оценку (score) каждому узлу. Оценки от разных плагинов суммируются. На этой фазе оцениваются доступные ресурсы на узлах: `pod capacity`, `affinity`, `volume provisioning` и другие. По итогам этой фазы выбирается узел с наибольшей оценкой. Если сразу несколько узлов получили максимальную оценку, узел выбирается случайным образом.

В итоге под запускается на выбранном узле.

Документация

- Общее описание scheduler.
- Система плагинов.
- Подробности фильтрации узлов.
- Исходный код scheduler.

Как изменить или расширить логику работы планировщика

Для изменения логики работы планировщика можно использовать механизм плагинов расширения.

Каждый плагин представляет собой вебхук, отвечающий следующим требованиям:

- Использование TLS.
- Доступность через сервис внутри кластера.
- Поддержка стандартных Verbs (`filterVerb = filter`, `prioritizeVerb = prioritize`).
- Также, предполагается что все подключаемые плагины могут кэшировать информацию об узле (`nodeCacheCapable: true`).

Подключить `extender` можно при помощи ресурса `KubeSchedulerWebhookConfiguration`.



При использовании опции `failurePolicy: Fail`, в случае ошибки в работе вебхука планировщик Kubernetes прекратит свою работу, и новые поды не смогут быть запущены.

Как происходит ротация сертификатов kubelet?

В Deckhouse Kubernetes Platform ротация сертификатов kubelet происходит автоматически.

Kubelet использует клиентский TLS-сертификат (`/var/lib/kubelet/pki/kubelet-client-current.pem`), при помощи которого может запросить у `kube-apiserver` новый клиентский сертификат или новый серверный сертификат (`/var/lib/kubelet/pki/kubelet-server-current.pem`).

Когда до истечения времени жизни сертификата остается 5-10% (случайное значение из диапазона) времени, kubelet запрашивает у `kube-apiserver` новый сертификат. С описанием алгоритма можно ознакомиться в официальной документации Kubernetes.

Время жизни сертификатов

По умолчанию время жизни сертификатов равно 1 году (8760 часов). При необходимости это значение можно изменить с помощью аргумента `--cluster-signing-duration` в манифесте `/etc/kubernetes/manifests/kube-controller-manager.yaml`. Но чтобы kubelet успел установить сертификат до его истечения, рекомендуем устанавливать время жизни сертификатов более, чем 1 час.



Если истекло время жизни клиентского сертификата, то kubelet не сможет делать запросы к `kube-apiserver` и не сможет обновить сертификаты. В данном случае узел (Node) будет помечен как `NotReady` и пересоздан.

Особенности работы с серверными сертификатами kubelet в Deckhouse Kubernetes Platform

В Deckhouse Kubernetes Platform для запросов в kubelet API используются IP-адреса. Поэтому в конфигурации kubelet поля `tlsCertFile` и `tlsPrivateKeyFile` не указываются, а используется динамический сертификат, который kubelet генерирует самостоятельно. Также, из-за использования динамического сертификата, в Deckhouse Kubernetes Platform (в модуле `operator-trivy`) отключены проверки CIS benchmark `AVD-KCV-0088` и `AVD-KCV-0089`, которые отслеживают, были ли переданы аргументы `--tls-cert-file` и `--tls-private-key-file` для kubelet.

Информация о логике работы с серверными сертификатами в Kubernetes

В kubelet реализована следующая логика работы с серверными сертификатами:

- Если `tlsCertFile` и `tlsPrivateKeyFile` не пустые, то kubelet будет использовать их как сертификат и ключ по

умолчанию.

- При запросе клиента в kubelet API с указанием IP-адреса (например `https://10.1.1.2:10250/`), для установления соединения по TLS-протоколу будет использован закрытый ключ по умолчанию (`tlsPrivateKeyFile`). В данном случае ротация сертификатов не будет работать.
- При запросе клиента в kubelet API с указанием названия хоста (например `https://k8s-node:10250/`), для установления соединения по TLS-протоколу будет использован динамически сгенерированный закрытый ключ из директории `/var/lib/kubelet/pki/`. В данном случае ротация сертификатов будет работать.
- Если `tlsCertFile` и `tlsPrivateKeyFile` пустые, то для установления соединения по TLS-протоколу будет использован динамически сгенерированный закрытый ключ из директории `/var/lib/kubelet/pki/`. В данном случае ротация сертификатов будет работать.

Как вручную обновить сертификаты компонентов управляющего слоя?

Может возникнуть ситуация, когда master-узлы кластера находятся в выключенном состоянии долгое время. За это время может истечь срок действия сертификатов компонентов управляющего слоя. После включения узлов сертификаты не обновятся автоматически, поэтому это необходимо сделать вручную.

Обновление сертификатов компонентов управляющего слоя происходит с помощью утилиты `kubeadm`. Чтобы обновить сертификаты, выполните следующие действия на каждом master-узле:

1. Найдите утилиту `kubeadm` на master-узле и создайте символическую ссылку с помощью следующей команды:

```
ln -s $(find /var/lib/containerd -name kubeadm -type f -executable -print -quit) /usr/bin/kubeadm
```

2. Обновите сертификаты:

```
kubeadm certs renew all
```

Как защитить чувствительные поля кастомных ресурсов?

Для защиты чувствительных полей (паролей, токенов или ключей) в схемах ресурсов от несанкционированного доступа через API, хранения в etcd в незашифрованном виде и попадания в журнал аудита используйте feature gate `CRDSensitiveData` совместно с маркером схемы `x-kubernetes-sensitive-data`.

Чтобы включить защиту полей, выполните следующие действия:

1. Включите шифрование etcd с помощью параметра `apiserver.encryptionEnabled` в настройках модуля. Feature gate `CRDSensitiveData` включается автоматически одновременно с шифрованием — его не следует указывать вручную.

⚠ Включение параметра `apiserver.encryptionEnabled` необратимо и приводит к перезапуску `kube-apiserver`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true
```

2. Отметьте чувствительные поля в схеме ресурсов маркером `x-kubernetes-sensitive-data: true`.

Маркер должен находиться на поле типа `string`, `integer`, `number`, `boolean`, `object` или `array` (маркер на `object` или `array` делает чувствительным всё поддерево); также поддерживаются поля с `x-kubernetes-int-or-string: true`.

Маркер нельзя указывать на корне схемы (узел `openAPIV3Schema`) и внутри веток `anyOf`, `oneOf`, `allOf`, `not`.

3. Для пользователей и `ServiceAccount`, которым нужен доступ к полным данным, добавьте в правила RBAC разрешение на вложенный ресурс API `<resource>/sensitive`.

Применяемые меры защиты

Защита	Описание
Шифрование в etcd	Весь ресурс шифруется при хранении с помощью провайдера шифрования AES-CBC (аналогично Kubernetes Secrets).
Фильтрация полей в API	Чувствительные поля удаляются из ответов на запросы <code>get</code> , <code>list</code> и <code>watch</code> , если у вызывающей стороны нет прав на вложенный ресурс API <code><resource>/sensitive</code> .
Маскировка в журнале аудита	Значения чувствительных полей заменяются на <code>"*****"</code> во всех событиях аудита, независимо от прав RBAC и уровня аудита.

Полный пример конфигурации и результатов доступен в разделе «Примеры».

Примеры

Подключение внешнего плагина планировщика

Пример подключения внешнего плагина планировщика через вебхук.

```

apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
    timeoutSeconds: 5

```

Защита ресурсов с чувствительными полями

Далее показан пример конфигурации с защитой чувствительных полей ресурсов с помощью feature gate `CRDSensitiveData` и маркера схемы `x-kubernetes-sensitive-data`.

Инструкция по включению защиты доступна в разделе «FAQ».

1. Включение шифрования etcd с помощью параметра `apiserver.encryptionEnabled`. Это автоматически активирует feature gate `CRDSensitiveData` для `kube-apiserver`.

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true

```

2. Определение чувствительных полей в конфигурации.

Поля, помеченные маркером `x-kubernetes-sensitive-data: true`, будут шифроваться в etcd и удаляться из ответов API для вызывающих сторон без доступа к субресурсу `<resource>/sensitive`.

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: dbconfigs.example.com
spec:
  group: example.com
  scope: Namespaced
  names:
    plural: dbconfigs
    singular: dbconfig
    kind: DbConfig
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              host:
                type: string
              username:
                type: string
              password:
                type: string
              x-kubernetes-sensitive-data: true

```

3. Создание кастомного ресурса с заполненными значениями в чувствительных полях.

```

apiVersion: example.com/v1
kind: DbConfig
metadata:
  name: primary
  namespace: default
spec:
  host: db.example.com
  username: admin
  password: s3cr3t

```

После сохранения объект целиком шифруется в etcd, значение `password` маскируется в журнале аудита и удаляется из ответов API, если у вызывающей стороны нет доступа к субресурсу `dbconfigs/sensitive`.

4. Настройка доступа к чувствительным полям с помощью RBAC через субресурс `<resource>/sensitive`.

```

# Обычная роль: может читать ресурс, но чувствительные поля будут удалены из ответа.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
---
# Привилегированная роль: может читать полные данные, включая чувствительные поля.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-sensitive-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["example.com"]
  resources: ["dbconfigs/sensitive"]
  verbs: ["get", "list", "watch"]

```

5. Результат защиты чувствительных полей.

- Пользователь с ролью `dbconfig-reader`, выполнивший команду `d8 k get dbconfig primary -o json`, увидит ресурс с удалёнными чувствительными полями:

```

{
  "spec": {
    "host": "db.example.com",
    "username": "admin"
  }
}

```

- Пользователь с ролью `dbconfig-sensitive-reader` увидит полные данные:

```

{
  "spec": {
    "host": "db.example.com",
    "username": "admin",
    "password": "s3cr3t"
  }
}

```

- В журнале аудита значения чувствительных полей всегда маскируются, независимо от прав вызывающей стороны:

```

{
  "spec": {
    "host": "db.example.com",
    "username": "admin",
    "password": "*****"
  }
}

```

Модуль csi-vsphere



Стадия жизненного цикла модуля: `Deprecated`

Модуль `csi-vsphere` предоставляет поддержку Container Storage Interface (CSI) для сред VMware vSphere, обеспечивая динамическое предоставление и управление постоянными томами хранения в кластерах Kubernetes, работающих на инфраструктуре vSphere.

i Используйте модуль `csi-vsphere`, когда в кластере отсутствует возможность применять модуль `cloud-provider-vsphere`, но требуется динамический заказ томов в инфраструктуре vSphere.

Этот модуль специально разработан для **статических кластеров Kubernetes** (не облачных), развернутых на VMware vSphere, и работает независимо от модуля `cloud-provider-vsphere`. Он интегрируется с `datastores vSphere` для предоставления возможностей постоянного хранения через CSI-драйвер vSphere.

Основные возможности

- Обеспечивает динамический заказ постоянных томов для статических кластеров Kubernetes на VMware vSphere.
- Автоматически обнаруживает `datastores vSphere` и создаёт ресурсы `StorageClass`.
- Поддерживает топологически осведомлённое размещение томов с помощью тегов регионов и зон vSphere.
- Обрабатывает создание, удаление, присоединение, отсоединение томов и онлайн-изменение размера (режим по умолчанию).
- Поддерживает режимы совместимости `Legacy`, `Default` и `Migration` для плавного перехода между версиями драйвера.

Модуль разворачивает компоненты VMware vSphere CSI-драйвера, которые:

- **Автоматически обнаруживают `datastores vSphere`** — читает данные обнаружения из секрета `d8-cloud-provider-discovery-data` для идентификации доступных `datastores` и их топологии.
- **Создают `StorageClass`** — автоматически генерирует ресурсы `StorageClass` для каждого обнаруженного `datastore`, отфильтрованные по списку исключений при настройке.
- **Предоставляют постоянные тома** — динамически создает постоянные тома на `datastores vSphere` при создании `PersistentVolumeClaim`.
- **Поддерживают операции с томами** — обрабатывает создание, удаление, присоединение, отсоединение томов и онлайн-изменение размера (в Режиме по умолчанию).
- **Реализуют топологическую осведомленность** — использует лейблы зон и регионов для правильного размещения томов в соответствии с топологией кластера vSphere.

Системные требования

Для использования модуля в кластере необходимо соблюдение следующих системных требований:

- Все виртуальные машины кластера должны быть созданы с использованием инструментов vSphere.
- Имя виртуальной машины в vSphere должно **точно совпадать** с именем узла (`hostname`) в кластере Deckhouse Kubernetes Platform.
- В настройках каждой виртуальной машины необходимо включить параметр `disk.EnableUUID:TRUE`. Он обеспечивает корректную работу модуля с дисковыми ресурсами и позволяет DKP идентифицировать подключенные тома.

Архитектура

CSI-драйвер состоит из следующих компонентов:

1. **CSI Controller** (работает на master-узлах):
 - **`vsphere-csi-controller`** — основной контроллер, управляющий жизненным циклом томов.
 - **`csi-provisioner`** — следит за PVC и запускает создание томов.
 - **`csi-attacher`** — обрабатывает присоединение томов к узлам.
 - **`csi-resizer`** — управляет онлайн-расширением томов.
 - **`vsphere-syncer`** — синхронизирует метаданные между Kubernetes и vSphere каждые 30 минут.
 - **`liveness-probe`** — мониторит здоровье контроллера.
2. **CSI Node** (DaemonSet на всех узлах):
 - **`vsphere-csi-node`** — узловой CSI-драйвер, обрабатывающий монтирование/размонтирование томов.
 - **`node-driver-registrar`** — регистрирует CSI-драйвер в kubelet.
 - **`liveness-probe`** — мониторит здоровье узлового драйвера.

Режимы совместимости

Модуль поддерживает следующие режимы работы (режим устанавливается через параметр `compatibilityFlag`):

- **Режим по умолчанию** (используется по умолчанию, `compatibilityFlag` не установлен или пустой):
 - использует текущий CSI-драйвер vSphere (`csi.vsphere.vmware.com`),
 - поддерживает CNS (Cloud Native Storage) тома,
 - есть поддержка онлайн-изменения размера томов,
 - рекомендуется для новых развертываний.
- **Устаревший режим** (`compatibilityFlag: "Legacy"`):
 - использует старый CSI-драйвер vSphere (`vsphere.csi.vmware.com`),
 - работает только с FCD (First Class Disk) томами,
 - нет поддержки онлайн-изменения размера,
 - используется для обратной совместимости.
- **Режим миграции** (`compatibilityFlag: "Migration"`):
 - использует обе версии драйвера одновременно,
 - облегчает миграцию из устаревшего (Legacy) в Режим по умолчанию (Default),
 - позволяет постепенный переход рабочих нагрузок.

Обнаружение и предоставление хранения

Модуль автоматически:

1. **Обнаруживает `datastores`** из данных обнаружения облачного провайдера.
2. **Генерирует имена `StorageClass`** путем очистки имен `datastores` для соответствия требованиям именования DNS Kubernetes.
3. **Фильтрует исключенное хранилище** на основе конфигурации `storageClass.exclude` (поддерживает regex-шаблоны).
4. **Создает `StorageClass`** с соответствующими параметрами:
 - `allowVolumeExpansion: true` (только Режим по умолчанию),
 - `volumeBindingMode: WaitForFirstConsumer` — гарантирует создание томов в той же зоне, в которой создаются поды;
 - топологические ограничения, соответствующие регионам и зонам vSphere.

Настройка `StorageClass` по умолчанию

Модуль автоматически создает `StorageClass` для каждого `Datastore` и `DatastoreCluster` из зон.

Чтобы задать `StorageClass` по умолчанию для кластера, используйте глобальный параметр `defaultClusterStorageClass`. Чтобы отфильтровать ненужные объекты `StorageClass`, задайте шаблоны исключений в параметре `storageClass.exclude`.

Жизненный цикл томов

Когда под запрашивает хранилище:

1. Создается PVC, ссылающийся на StorageClass, управляемый модулем.
2. CSI provisioner создает том на указанном datastore vSphere.
3. CSI attacher присоединяет том к соответствующему узлу.
4. CSI node driver монтирует том в под.
5. vsphere-synsger периодически синхронизирует метаданные для обеспечения согласованности.

Ограничения

- Модуль разработан исключительно для статических (не облачных) кластеров Kubernetes на vSphere.
- Устаревший режим не поддерживает онлайн-изменение размера томов.
- Расширение томов требует текущего CSI-драйвера (`csi.vsphere.vmware.com`).
- Объекты StorageClass управляются автоматически — ручные изменения могут быть перезаписаны.

Настройки

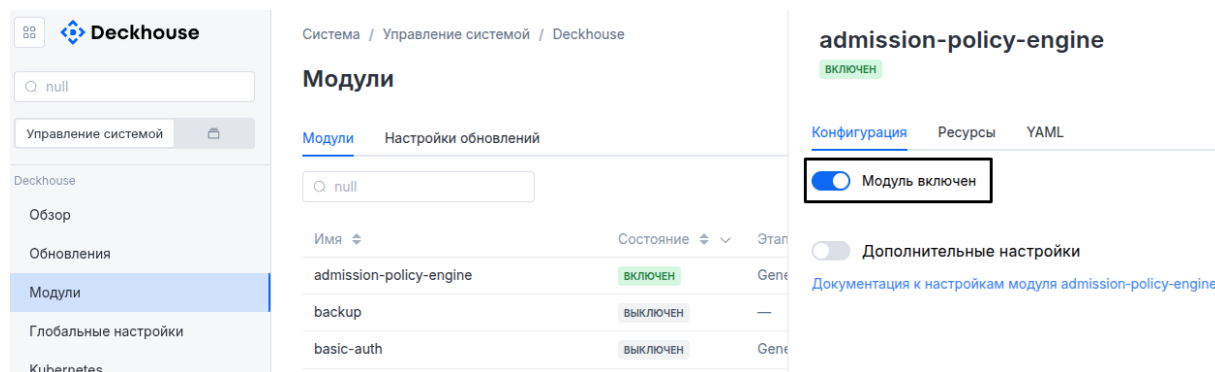
✖ Стадия жизненного цикла модуля: Deprecated

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `csi-vsphere`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `csi-vsphere`:

```
d8 system module enable csi-vsphere
```

- **С помощью ModuleConfig `csi-vsphere`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `csi-vsphere` (создайте его, при необходимости).

Пример манифеста для включения модуля `csi-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-vsphere
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `csi-vsphere`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация

Ресурсы

YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация

YAML

Вы также можете отредактировать объект ModuleConfig `csi-vsphere` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `csi-vsphere`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `csi-vsphere` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc csi-vsphere
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `csi-vsphere`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `csi-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-vsphere
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.disableTimesync**
булевый

Отключение синхронизации времени со стороны vSphere.

i

Внимание. Данный параметр не отключает NTP-демоны в гостевой ОС, а лишь отключает корректировку времени со стороны ESXi.

- **settings.host**
строка

Обязательный параметр

Доменное имя сервера vCenter.

Пример:

```
host: myhost
```

- **settings.insecure**
булевый

Установите `true`, если vCenter использует самоподписанный сертификат.

- **settings.password**
строка

Обязательный параметр

Пароль.

Пример:

```
password: myPaSsWd
```

- **settings.region**

строка

Обязательный параметр

Тег, прикрепленный к vSphere Datacenter, в котором будут происходить все операции: заказ виртуальных машин, размещение их дисков на datastore, подключение к сети.

Пример:

```
region: myreg
```

- **settings.regionTagCategory**

строка

Обязательный параметр

Имя категории тегов, используемых для идентификации региона (vSphere Datacenter).

Пример:

```
regionTagCategory: myregtagcat
```

- **settings.storageClass**

объект

- **settings.storageClass.compatibilityFlag**

строка

Флаг, разрешающий использование старой версии CSI:

- **Legacy** — используется старая версия драйвера — только диски FCD, без онлайн-изменения размера;
- **Migration** — оба драйвера будут доступны в кластере одновременно. Этот режим используется для миграции со старого драйвера.

Допустимые значения: **Legacy**, **Migration**

- **settings.storageClass.exclude**

массив строк

Полные имена (или regex выражения имен) StorageClass'ов, которые не будут созданы в кластере.

Пример:

```
exclude:
- ".*-lun101-.*"
- slow-lun103-1c280603
```

- **settings.username**

строка

Обязательный параметр

Логин.

Пример:

```
username: myuname
```

- **settings.vmFolderPath**

строка

Обязательный параметр

Путь до VirtualMachine Folder, в котором будут создаваться клонированные виртуальные машины.

Пример:

```
vmFolderPath: dev/test
```

- **settings.zoneTagCategory**

строка

Обязательный параметр

Имя категории тегов, используемых для идентификации зоны (vSphere Cluster).

Пример:

```
zoneTagCategory: myzonetagcat
```

- **settings.zones**

массив строк

Обязательный параметр

Глобальное ограничение набора зон, с которыми работает данный cloud-провайдер.

Пример:

```
zones:
- zonea
- zoneb
```

Подготовка окружения

✖ Стадия жизненного цикла модуля: Deprecated

В этом разделе описана подготовка vCenter и vSphere к работе модуля `csi-vsphere`.

Список необходимых ресурсов

Для работы модуля требуется наличие следующих ресурсов:

- User с необходимым набором прав.
- Network с DHCP и доступом в интернет.
- Datacenter с соответствующим тегом `k8s-region`.
- Cluster с соответствующим тегом `k8s-zone`.
- Datastore в любом количестве с соответствующими тегами.

Установка govc

Для дальнейшей конфигурации `csi-vsphere` понадобится vSphere CLI — `govc`.

После установки задайте переменные окружения для работы с vCenter:

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Создание тегов и категорий тегов

В `csi-vsphere` нет понятий «регион» и «зона». «Регионом» в `csi-vsphere` является Datacenter, а «зоной» — Cluster. Связь между этими объектами задаётся через теги.

Чтобы связать объекты Cluster и Datacenter, выполните следующие действия:

1. Создайте категории тегов:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

2. Создайте теги в каждой категории. Если планируется несколько зон (Cluster), создайте отдельный тег для каждой:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

3. Назначьте тег региона на Datacenter:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>
```

4. Назначьте теги зон на объекты Cluster:

```
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
```

Конфигурация Datastore

i Для динамического заказа PersistentVolume необходимо, чтобы Datastore был доступен на каждом хосте ESXi (shared datastore).

Для автоматического создания StorageClass в кластере назначьте теги региона и зоны на объекты Datastore:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Создание и назначение роли

i Ввиду разнообразия подключаемых к `csi-vsphere` SSO-провайдеров шаги по созданию пользователя в данном разделе не рассматриваются.

Предлагаемая ниже роль включает права, достаточные для всех компонентов DKP. Детальный перечень привилегий описан в документации модуля `cloud-provider-vsphere`.

Создайте роль с необходимыми правами:

```
govc role.create deckhouse \
  Cns.Searchable Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
  Global.GlobalTag Global.SystemTag Network.Assign StorageProfile.View \
  $(govc role.ls Admin | grep -F -e 'Folder.' -e 'InventoryService.' -e 'Resource.' -e 'VirtualMachine.')
```

Назначьте пользователю роль на объекте vCenter:

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

Модуль deckhouse

Этот модуль настраивает в Deckhouse:

- **Уровень логирования**
- **Набор модулей, включенных по умолчанию**

Обычно используется набор модулей `Default`, который подходит в большинстве случаев.

Независимо от используемого набора включенных по умолчанию модулей любой модуль может быть явно включен или выключен в конфигурации Deckhouse (подробнее про включение и отключение модуля).

- **Канал обновлений**

В Deckhouse реализован механизм автоматического обновления. Этот механизм использует 5 каналов обновлений, различающиеся стабильностью и частотой выхода версий. Ознакомьтесь подробнее с тем, как работает механизм автоматического обновления и как установить желаемый канал обновлений.

- **Режим обновлений и окна обновлений**

Deckhouse может использовать **ручной** или **автоматический** режим обновлений.

В ручном режиме обновлений автоматически применяются только важные исправления (patch-релизы), и для перехода на новый релиз Deckhouse требуется ручное подтверждение.

В автоматическом режиме обновлений, если в кластере **не установлены** окна обновлений, переход на новый релиз Deckhouse осуществляется сразу после его появления на соответствующем канале обновлений. Если же в кластере **установлены** окна обновлений, переход на более свежий релиз Deckhouse начнется в ближайшее доступное окно обновлений после появления новой версии на канале обновлений.

- **Сервис валидирования кастомных ресурсов**

Сервис валидирования предотвращает создание кастомных ресурсов с некорректными данными или внесение таких данных в уже существующие кастомные ресурсы. Отслеживаются только ресурсы, находящиеся под управлением модулей Deckhouse.

Обновление релизов Deckhouse

Просмотр статуса релизов Deckhouse

Список последних релизов в кластере можно получить командой `d8 k get deckhouse-releases`. По умолчанию хранятся 10 последних релизов и все будущие. Каждый релиз может иметь один из следующих статусов:

- **Pending** — релиз находится в ожидании, ждет окна обновления, настроек канареечного развертывания и т. д. Подробности можно увидеть с помощью команды `d8 k describe deckhouse-release $name`.
- **Deployed** — релиз применен. Это значит, что образ пода Deckhouse уже поменялся на новую версию, но при этом процесс обновления всех компонентов кластера идет асинхронно, так как зависит от многих настроек.
- **Superseded** — релиз устарел и больше не используется.
- **Suspended** — релиз отменен (например, в нем обнаружилась ошибка). Релиз переходит в этот статус, если его отменили и при этом он еще не был применен в кластере.

Процесс обновления

В момент перехода в статус `Deployed` релиз меняет версию (tag) образа Deckhouse. После запуска Deckhouse начнет проверку и обновление всех модулей, которые поменялись с предыдущего релиза. Длительность обновления зависит от настроек и размера кластера. Например, если у вас много `NodeGroup`, они будут обновляться продолжительное время, если много `IngressNginxController` — они будут обновляться по одному и это тоже займет некоторое время.

Ручное применение релизов

Если выбран ручной режим обновления и скопилось несколько релизов, их можно сразу одобрить к применению. В этом случае Deckhouse будет обновляться последовательно, сохраняя порядок релизов и меняя статус каждого примененного релиза.

Закрепление релиза

Под **закреплением** релиза подразумевается полное или частичное отключение автоматического обновления версий Deckhouse.

Есть три варианта ограничения автоматического обновления Deckhouse:

- Установить ручной режим обновления.

В этом случае вы остановитесь на текущей версии, сможете получать обновления в кластер, но для применения обновления необходимо будет выполнить ручное действие. Это относится и к patch-версиям, и к минорным версиям.

Для установки ручного режима обновления необходимо в `ModuleConfig` `deckhouse` установить параметр `settings.update.mode` в `Manual`:

```
d8 k patch mc deckhouse --type=merge -p='{ "spec": { "settings": { "update": { "mode": "Manual" } } } }'
```

- Установить режим автоматического обновления для патч-версий.

В этом случае вы остановитесь на текущем релизе, но будете получать patch-версии текущего релиза (с учетом установленных окон обновлений). Для применения обновления минорной версии релиза необходимо будет выполнить ручное действие.

Например: текущая версия DKP `v1.70.1`, после установки режима автоматического обновления для патч-версий, Deckhouse сможет обновиться до версии `v1.70.1`, но не будет обновляться до версии `v1.71.*` и выше.

Для установки режима автоматического обновления для патч-версий необходимо в ModuleConfig `deckhouse` установить параметр `settings.update.mode` в `AutoPatch`:

```
d8 k patch mc deckhouse --type=merge -p='{ "spec": { "settings": { "update": { "mode": "AutoPatch" } } } }'
```

- Установить конкретный тег для Deployment `deckhouse` и удалить параметр `releaseChannel` из конфигурации модуля `deckhouse`.

В таком случае DKP останется на конкретной версии, никакой информации о новых доступных версиях (объекты `DeckhouseRelease`) в кластере появляться не будет.

Пример установки версии `v1.66.3` для DKP EE и удаления параметра `releaseChannel` из конфигурации модуля `deckhouse`:

```
d8 k -ti -n d8-system exec svc/deckhouse-leader -C deckhouse -- kubectl set image deployment/deckhouse
deckhouse=registry.deckhouse.ru/deckhouse/ee:v1.66.3
d8 k patch mc deckhouse --type=json -p='{ "op": "remove", "path": "/spec/settings/releaseChannel" }'
```

Priority Classes

Модуль создает в кластере набор классов приоритета (`PriorityClass`) и назначает их компонентам, установленным Deckhouse, и приложениям в кластере.

Функциональность классов приоритета реализуется планировщиком (`scheduler`), который позволяет учитывать приоритет пода (определяемый его принадлежностью к классу) при планировании.

Например, при разворачивании в кластере подов с `priorityClassName: production-low`, если в кластере не будет доступных ресурсов для данного пода, Kubernetes начнет вытеснять поды с наименьшим приоритетом. То есть сначала будут вытеснены все поды с `priorityClassName: develop`, затем — с `cluster-low` и так далее.

При указании класса приоритета очень важно понимать тип приложения и окружение, в котором оно будет работать. Указание любого класса приоритета не уменьшит его фактический приоритет, так как если у пода не установлен приоритет, то планировщик считает его самым низким.

Практические примеры создания и использования `PriorityClass`, демонстрация вытеснения и диагностика описаны в разделе «Примеры».



Нельзя использовать классы приоритета `system-node-critical`, `system-cluster-critical`, `cluster-medium`, `cluster-low`.

Устанавливаемые модулем классы приоритета (в порядке приоритета от высшего к низшему):

Класс приоритета	Описание	Значение
<code>system-node-critical</code>	Компоненты кластера, которые обязаны присутствовать на узле. Также полностью защищает от вытеснения kubelet'ом. Примеры: <code>node-exporter</code> , <code>csi</code> и другие.	2000001000
<code>system-cluster-critical</code>	Компоненты кластера, без которых его корректная работа невозможна. Этим <code>PriorityClass</code> ом обязательно помечаются <code>MutatingWebhooks</code> и <code>Extension API servers</code> . Также полностью защищает от вытеснения kubelet'ом. Примеры: <code>kube-dns</code> , <code>kube-proxy</code> , <code>cni-flannel</code> , <code>cni-cillium</code> и другие.	2000000000
<code>production-high</code>	Stateful-приложения, отсутствие которых в <code>production</code> -окружении приводит к полной недоступности сервиса или потере данных. Примеры: <code>PostgreSQL</code> , <code>Memcached</code> , <code>Redis</code> , <code>MongoDB</code> и другие.	9000
<code>cluster-medium</code>	Компоненты кластера, влияющие на мониторинг (алерты, диагностика) и автомасштабирование. Без мониторинга невозможно оценить масштабы происшествия, без автомасштабирования — предоставить приложениям необходимые ресурсы. Примеры: <code>deckhouse</code> , <code>node-local-dns</code> , <code>grafana</code> , <code>upmeter</code> и другие.	7000
<code>production-medium</code>	Основные stateless-приложения в <code>production</code> -окружении, которые отвечают за работу сервиса для посетителей.	6000
<code>deployment-machinery</code>	Компоненты кластера, используемые для сборки и деплоя в кластер.	5000
<code>production-low</code>	Приложения в <code>production</code> -окружении (cron-задания, административные панели, batch-процессы), без которых можно обойтись некоторое время. Если batch или cron-задачи нельзя прерывать, их следует отнести к <code>production-medium</code> .	4000
<code>staging</code>	Staging-окружения для приложений.	3000

Класс приоритета	Описание	Значение
cluster-low	Компоненты кластера, без которых эксплуатация возможна, но которые желательны. Примеры: dashboard, cert-manager, prometheus и другие.	2000
develop (по умолчанию)	Develop-окружения для приложений. Класс по умолчанию, если не указан иной класс.	1000
standby	Класс не предназначен для приложений. Используется в системных целях для резервирования узлов.	-1

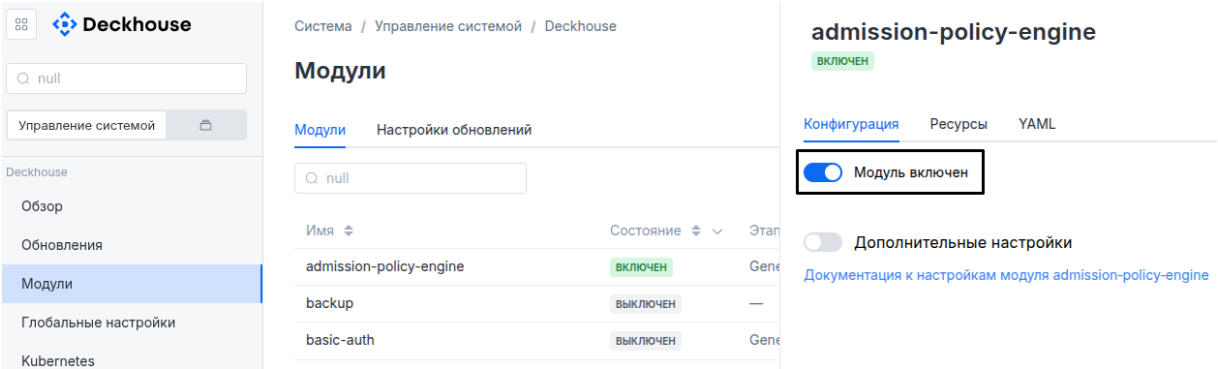
Настройки

Модуль **включен** по умолчанию в наборах модулей: Default, Managed, Minimal.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль deckhouse, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду d8 system module enable для включения модуля, или d8 system module disable для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля deckhouse:

```
d8 system module enable deckhouse
```

- **С помощью ModuleConfig deckhouse.**

Установите spec.enabled в true или false в ModuleConfig deckhouse (создайте его, при необходимости).

Пример манифеста для включения модуля deckhouse:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль deckhouse, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `deckhouse` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `deckhouse`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `deckhouse` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc deckhouse
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `deckhouse`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.allowExperimentalModules**
булевый

Разрешает использование экспериментальных модулей (модули на стадии жизненного цикла `Experimental`).

Экспериментальные модули могут быть нестабильными и не поддерживаются в рамках SLA Deckhouse. Используйте их с осторожностью.

По умолчанию: `false`

Пример:

```
allowExperimentalModules: false
```

- **settings.bundle**
строка

Набор модулей DKP, которые включаются по умолчанию.

Изменить значение параметра в уже развёрнутом кластере невозможно.

Возможные значения:

- `Default` — рекомендованный набор модулей для работы кластера. Включает модули для мониторинга, контроля авторизации, организации работы сети и других базовых потребностей платформы;
- `Managed` — набор модулей для `managed`-кластеров от облачных провайдеров;
- `Minimal` — минимально возможный набор, который включает единственный модуль (этот).

Внимание. В набор модулей `Minimal` не включён ряд базовых модулей платформы (например, модуль работы с CNI). Без включения базовых модулей DKP сможет работать только в уже развёрнутом кластере. Читайте подробнее в разделе «Особенности работы с набором модулей `Minimal`».

По умолчанию: `Default`

Допустимые значения: `Default`, `Minimal`, `Managed`

Пример:

```
bundle: Default
```

◦ **settings.highAvailability**

булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

◦ **settings.license**

объект

Настройки редакции Deckhouse Kubernetes Platform.

■ **settings.license.edition**

строка

Редакция Deckhouse Kubernetes Platform.

- `CE` — Community Edition.
- `BE` — Basic Edition.
- `EE` — Enterprise Edition.
- `SE` — Standard Edition.
- `SE-plus` — Standard Edition+.
- `CSE` — Certified Security Edition.

По умолчанию: `CE`

Пример:

```
edition: CE
```

◦ **settings.logLevel**

строка

Уровень логирования Deckhouse.

По умолчанию: `Info`

Допустимые значения: `Trace`, `Debug`, `Info`, `Error`

Пример:

```
logLevel: Info
```

◦ **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, `nodeSelector` будет определяться автоматически.

Внимание. Deckhouse перестанет работать, если в `nodeSelector` указать несуществующий лейбл. Для восстановления работоспособности необходимо изменить значения на правильные в `ModuleConfig/deckhouse` и `deployment/deckhouse`.

◦ **settings.registry**

объект

Настройки для доступа к хранилищу образов контейнеров (container registry) с образами Deckhouse.

Этот параметр позволяет настроить, как DKP будет подключаться к container registry для загрузки образов своих компонентов. Настройки обрабатываются внутренним модулем `registry`. В кластерах, не поддерживающих данную конфигурацию, настройка подключения к registry выполняется вручную с помощью вспомогательного скрипта `change-registry`.

Примеры:

```
registry:
  mode: Unmanaged
```

```
registry:
  mode: Unmanaged
  unmanaged:
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    license: DECKHOUSE_LICENSE_KEY
```

```
registry:
  mode: Direct
  direct:
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    license: DECKHOUSE_LICENSE_KEY
```

```
registry:
  mode: Proxy
  proxy:
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    license: DECKHOUSE_LICENSE_KEY
```

```
registry:
  mode: Local
```

■ **settings.registry.direct** объект

Параметры для режима доступа `Direct`. Эти параметры обязательны, если `mode` установлен в `Direct`.

Примеры:

```
direct:
  imagesRepo: registry.deckhouse.io/deckhouse/ee
  license: DECKHOUSE_LICENSE_KEY
```

```
direct:
  imagesRepo: my-private-registry.com/deckhouse
  username: user
  password: password
  scheme: HTTPS
  ca: |
    -----BEGIN CERTIFICATE-----
    MIIC...
    -----END CERTIFICATE-----
```

■ **settings.registry.direct.ca** строка

Корневой сертификат центра сертификации (CA) в формате PEM для проверки серверного сертификата container registry.

Параметр необходимо указывать, если container registry использует самоподписанный сертификат или сертификат, выданный частным центром сертификации.

Если параметр не указан, будут использоваться сертификаты, определенные в хранилище корневых сертификатов узла.

■ **settings.registry.direct.checkMode** строка

Режим проверки container registry:

- `Default` (по умолчанию) — проверяет наличие всех критически важных компонентов в container registry.
- `Relax` — проверяет только наличие тега образа Deckhouse в container registry (используется для переключения между редакциями Deckhouse).

По умолчанию: `Default`

Допустимые значения: `Default`, `Relax`

■ **settings.registry.direct.imagesRepo** строка

Обязательный параметр

Адрес репозитория в container registry, где хранятся образы Deckhouse. В изолированных средах это может быть адрес частного registry, который зеркалирует публичный.

По умолчанию: `registry.deckhouse.io/deckhouse/ce`

Шаблон: `^[0-9a-zA-Z\.\-]+\(:[0-9]{1,5}\)?(\/[0-9a-zA-Z\.\-\\_]+)*$`

Примеры:

```
imagesRepo: registry.deckhouse.io/deckhouse/ee
```

```
imagesRepo: private-registry:5000/deckhouse/ce
```

■ **settings.registry.direct.license** строка

Лицензионный ключ для доступа к container registry Deckhouse. Этот ключ используется в качестве пароля с именем пользователя `license-token`. Это поле является взаимоисключающим с `username/password`.

Пример:

```
license: DECKHOUSE_LICENSE_KEY
```

■ **settings.registry.direct.password** строка

Пароль для аутентификации в container registry.

Параметр обязателен, если не указан параметр `license`.

Пример:

```
password: registry-password
```

■ settings.registry.direct.scheme

строка

Протокол для подключения к container registry.

Используйте `HTTP` только для небезопасных, доверенных container registry (например, в изолированной среде). По умолчанию используется `HTTPS`.

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

Пример:

```
scheme: HTTPS
```

■ settings.registry.direct.username

строка

Имя пользователя для аутентификации в реестре контейнеров. Это поле обязательно, если не предоставлен `license`.

Пример:

```
username: registry-user
```

■ settings.registry.mode

строка

Режим доступа к container registry с образами Deckhouse.

- `Unmanaged` — режим по умолчанию. В этом режиме модуль `registry` не запускает внутренний registry. Для настройки необходимо задать параметры в секции `unmanaged`. **Важно.** Изменение registry в `Unmanaged` режиме приведёт к перезапуску всех компонентов DKP.
- `Direct` — режим прямого доступа к registry. Используется «виртуальный» адрес registry, который «на лету» подменяется на реальный. Это позволяет изменять registry без полного перезапуска компонентов. **Важно.** При первом переключении с режима `Unmanaged` на режим `Direct` произойдёт полный перезапуск всех компонентов DKP.
- `Proxy` — режим проксирующего registry. Запускает статичные поды с кеширующим прокси registry. Режим позволяет сократить количество запросов к внешнему registry за счёт кеширования образов. Кеш образов хранится на master-узлах. Доступ к проксирующему registry — по фиксированному адресу, аналогично режиму `Direct`. **Важно.** Поддерживается только на статичных кластерах. При первом переключении с режима `Unmanaged` на режим `Proxy` произойдёт полный перезапуск всех компонентов DKP.
- `Local` — режим локального registry. Запускает статичные поды с локальным registry. Режим позволяет кластеру работать в изолированной среде. Данные хранятся на master-узлах. Доступ к registry — по фиксированному адресу, аналогично режиму `Direct`. **Важно.** Поддерживается только на статичных кластерах. При первом переключении с режима `Unmanaged` на режим `Local` произойдёт полный перезапуск всех компонентов DKP.

По умолчанию: `Unmanaged`

Допустимые значения: `Unmanaged`, `Direct`, `Proxy`, `Local`

■ settings.registry.proxy

объект

Параметры для режима доступа `Proxy`. Эти параметры обязательны, если `mode` установлен в `Proxy`.

■ settings.registry.proxy.ca

строка

Корневой сертификат центра сертификации (CA) в формате PEM для проверки серверного сертификата container registry.

Параметр необходимо указывать, если container registry использует самоподписанный сертификат или сертификат, выданный частным центром сертификации.

Если параметр не указан, будут использоваться сертификаты, определенные в хранилище корневых сертификатов узла.

■ settings.registry.proxy.checkMode

строка

Режим проверки container registry:

- `Default` (по умолчанию) — проверяет наличие всех критически важных компонентов в container registry.
- `Relax` — проверяет только наличие тега образа Deckhouse в container registry (используется для переключения между редакциями Deckhouse).

По умолчанию: `Default`

Допустимые значения: `Default`, `Relax`

■ settings.registry.proxy.imagesRepo

строка

Обязательный параметр

Адрес репозитория в container registry, где хранятся образы Deckhouse. В изолированных средах это может быть адрес частного registry, который зеркалирует публичный.

По умолчанию: `registry.deckhouse.io/deckhouse/ce`

Шаблон: `^[0-9a-zA-Z\.\-]+\(:[0-9]{1,5}\)?(\/[0-9a-zA-Z\.\-_\-]+)*$`

Примеры:

```
imagesRepo: registry.deckhouse.io/deckhouse/ee
```

```
imagesRepo: private-registry:5000/deckhouse/ce
```

■ **settings.registry.proxy.license**

строка

Лицензионный ключ для доступа к container registry Deckhouse. Этот ключ используется в качестве пароля с именем пользователя `license-token`. Это поле является взаимоисключающим с `username/password`.

Пример:

```
license: DECKHOUSE_LICENSE_KEY
```

■ **settings.registry.proxy.password**

строка

Пароль для аутентификации в container registry.

Параметр обязателен, если не указан параметр `license`.

Пример:

```
password: registry-password
```

■ **settings.registry.proxy.scheme**

строка

Протокол для подключения к container registry.

Используйте `HTTP` только для небезопасных, доверенных container registry (например, в изолированной среде). По умолчанию используется `HTTPS`.

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

Пример:

```
scheme: HTTPS
```

■ **settings.registry.proxy.ttl**

строка

TTL определяет время жизни кешируемых образов. По истечении TTL образы будут автоматически удалены из кеша. Если значение не задано, TTL по умолчанию составляет 7 дней (168 часов).

Поддерживаемые единицы измерения: `h` (часы), `m` (минуты), `s` (секунды). Минимальное допустимое значение — 5 минут.

Шаблон: `^(?: (\d+)h)?(?: (\d+)m)?(?: (\d+)s)?$`

■ **settings.registry.proxy.username**

строка

Имя пользователя для аутентификации в реестре контейнеров. Это поле обязательно, если не предоставлен `license`.

Пример:

```
username: registry-user
```

■ **settings.registry.unmanaged**

объект

Параметры для режима доступа `Unmanaged`.

Примеры:

```
unmanaged:
  imagesRepo: registry.deckhouse.io/deckhouse/ee
  license: DECKHOUSE_LICENSE_KEY
```

```
unmanaged:
  imagesRepo: my-private-registry.com/deckhouse
  username: user
  password: password
  scheme: HTTPS
  ca: |
    -----BEGIN CERTIFICATE-----
    MIIC...
    -----END CERTIFICATE-----
```

■ **settings.registry.unmanaged.ca**

строка

Корневой сертификат центра сертификации (CA) в формате PEM для проверки серверного сертификата container registry.

Параметр необходимо указывать, если container registry использует самоподписанный сертификат или сертификат, выданный частным центром сертификации.

Если параметр не указан, будут использоваться сертификаты, определенные в хранилище корневых

сертификатов узла.

■ **settings.registry.unmanaged.checkMode**

строка

Режим проверки container registry:

- **Default** (по умолчанию) — проверяет наличие всех критически важных компонентов в container registry.
- **Relax** — проверяет только наличие тега образа Deckhouse в container registry (используется для переключения между редакциями Deckhouse).

По умолчанию: `Default`

Допустимые значения: `Default`, `Relax`

■ **settings.registry.unmanaged.imagesRepo**

строка

Обязательный параметр

Адрес репозитория в container registry, где хранятся образы Deckhouse. В изолированных средах это может быть адрес частного registry, который зеркалирует публичный.

По умолчанию: `registry.deckhouse.io/deckhouse/ce`

Шаблон: `^[0-9a-zA-Z\.\-]+\(:[0-9]{1,5}\)?(\/[0-9a-zA-Z\.\-\_]+)*$`

Примеры:

```
imagesRepo: registry.deckhouse.io/deckhouse/ee
```

```
imagesRepo: private-registry:5000/deckhouse/ce
```

■ **settings.registry.unmanaged.license**

строка

Лицензионный ключ для доступа к container registry Deckhouse. Этот ключ используется в качестве пароля с именем пользователя `license-token`. Это поле является взаимоисключающим с `username/password`.

Пример:

```
license: DECKHOUSE_LICENSE_KEY
```

■ **settings.registry.unmanaged.password**

строка

Пароль для аутентификации в container registry.

Параметр обязателен, если не указан параметр `license`.

Пример:

```
password: registry-password
```

■ **settings.registry.unmanaged.scheme**

строка

Протокол для подключения к container registry.

Используйте `HTTP` только для небезопасных, доверенных container registry (например, в изолированной среде). По умолчанию используется `HTTPS`.

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

Пример:

```
scheme: HTTPS
```

■ **settings.registry.unmanaged.username**

строка

Имя пользователя для аутентификации в реестре контейнеров. Это поле обязательно, если не предоставлен `license`.

Пример:

```
username: registry-user
```

○ **settings.releaseChannel**

строка

Желаемый канал обновлений Deckhouse (переключение на него произойдет при появлении такой возможности).

Порядок возрастания стабильности канала обновлений (от **наименее** стабильного к **наиболее** стабильному): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `LTS`.

Допустимые значения: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `LTS`

Пример:

```
releaseChannel: Stable
```

○ **settings.reservedPublicHosts**

объект

Настройки резервирования имён хостов веб-интерфейсов Deckhouse, которое запрещает занимать их приложениям.

Поле принимается, чтобы ModuleConfig, в котором оно уже записано, продолжал проходить валидацию. Admission в этом патче выключен: политики резервировали любое имя, которое способен породить часть значений `publicDomainTemplate`, и из-за этого отклонялись Ingress приложений. Настройки ниже игнорируются до релиза, в котором резервирование снова будет применяться. Значение `mode` по умолчанию — `List`, чтобы при возврате admission кластер, в котором поле не задавали, не оказался снова в режиме `Template`.

Deckhouse формирует эти имена хостов из глобального параметра `global.modules.publicDomainTemplate` и резервирует их в масштабе кластера: ресурс `Ingress`, а также `Gateway`, `GRPCRoute`, `HTTPRoute`, `ListenerSet` или `TLSRoute` из `Gateway API`, созданный вне системного пространства имён, не может указать ни одно из них. Эти параметры изменяют набор зарезервированных имён в обе стороны.

Резервирование действует для отдельного имени хоста во всех пространствах имён. Чтобы вместо этого исключить из резервирования пространство имён целиком, установите на него лейбл `security.deckhouse.io/reserved-hosts-bypass: "true"`.

Пример:

```
reservedPublicHosts:
  mode: List
  additionalHosts:
    - admin.example.com
  excludedServices:
    - grafana
```

■ `settings.reservedPublicHosts.additionalHosts` массив строк

Имена хостов, которые нужно зарезервировать дополнительно к тем, которые публикует Deckhouse.

Каждый элемент — полное имя хоста. Сравнение точное и без учёта регистра, как и для встроенных имён. Wildcard-имена не поддерживаются.

Элемент всегда резервирует имя хоста, независимо от значения параметра `excludedServices`, поэтому два параметра не могут противоречить друг другу.

По умолчанию: `[]`

Пример:

```
additionalHosts:
  - admin.example.com
```

■ Элемент массива строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ `settings.reservedPublicHosts.excludedServices` массив строк

Имена сервисов, имя хоста которых не нужно резервировать.

Используйте параметр в кластере, где приложение вне системного пространства имён обоснованно обслуживает путь под одним из таких имён хостов.

Имя — это та часть, которую подставляет `global.modules.publicDomainTemplate`, например `grafana` для `grafana.example.com`, а не само имя хоста. Имя хоста, в которое оно разворачивается, публикуется в ключе `excludedHosts` ConfigMap `d8-reserved-public-hosts` в пространстве имён `d8-system` — там можно проверить, что имя сделало то, что предполагалось.

В режиме `mode: Template` работает любое имя, поскольку резервирование покрывает весь домен, а не только то, что публикует Deckhouse. В режиме `mode: List` действует только имя, которое Deckhouse действительно публикует, потому что только такие имена там зарезервированы; остальные перечисляются в ключе `unknownExcludedServices` того же ConfigMap — туда стоит смотреть, если имя хоста неожиданно осталось зарезервированным.

Исключение имени снимает резервирование этого имени хоста во всех пространствах имён, включая те, в которых Deckhouse сам его обслуживает.

По умолчанию: `[]`

Пример:

```
excludedServices:
  - grafana
```

■ Элемент массива строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

■ `settings.reservedPublicHosts.mode` строка

Какая часть домена платформы резервируется.

По умолчанию — `List`.

- `Template` — все имена хостов, которые может сформировать `global.modules.publicDomainTemplate`, а также wildcard-форма шаблона, если `%s` занимает первую метку целиком. Модулю не нужно просить о включении в резервирование: имя хоста зарезервировано с того момента, как шаблон способен его сформировать, независимо от того, публикует ли его сегодня какой-либо модуль Deckhouse. Имя хоста, которое обоснованно обслуживает приложение, возвращается через параметр `excludedServices` ниже либо через лейбл на пространстве имён. Исключение — wildcard-форма: она резервируется точным совпадением, и `excludedServices` её не освобождает, поэтому приложению, которое её обслуживает, нужен лейбл на

пространстве имён или режим `List` ниже.

- `List` — только имена хостов тех сервисов Deckhouse, о публикации которых Deckhouse знает. Резервирование уже, и публичный домен, который Deckhouse начнёт публиковать позже, остаётся незарезервированным, пока список не будет обновлён.

Имена хостов, которые приложения уже обслуживают, однократно записываются в момент, когда режим `Template` начинает действовать, и остаются разрешёнными. При переключении с `List` на `Template` запись создаётся тогда же, поэтому приложение, появившееся в период действия режима `List`, тоже попадает в неё.

Запись создаётся один раз и повторно не создаётся. При переключении на `List` и обратно на `Template` сохраняется запись, сделанная в первый раз, поэтому приложение, занявшее зарезервированное имя хоста в промежутке, будет отклонено при следующей записи. Повторное создание записи при каждом возврате позволило бы обойти резервирование: переключиться на `List`, занять имя хоста и переключиться обратно.

На приложения экосистемы это не влияет ни в одном из режимов: `global.applications.publicDomainTemplate` размещает имя приложения и его пространство имён в двух разных метках, поэтому в имени хоста приложения на одну метку больше, чем в любом имени, которое покрывает это резервирование.

По умолчанию: `List`

Допустимые значения: `Template`, `List`

Примеры:

```
mode: Template
```

```
mode: List
```

◦ **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, `tolerations` будет определяться автоматически.

Внимание. Deckhouse перестанет работать, если указать некорректные `tolerations`. Для восстановления работоспособности необходимо изменить значения на корректные в `ModuleConfig/deckhouse` и `deployment/deckhouse`.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

◦ **settings.update**

объект

Настройки режима и окон обновления Deckhouse.

Пример:

```
update:
  windows:
    - from: '8:00'
      to: '15:00'
  days:
    - Tue
    - Sat
  disruptionApprovalMode: Manual
  notification:
    webhook: https://release-webhook.mydomain.com
    minimalNotificationTime: 6h
  auth:
    basic:
      username: user
      password: password
```

■ **settings.update.disruptionApprovalMode**

Параметр устарел
строка

Режим обновления версий Deckhouse, имеющих изменения, которые могут привести к кратковременному простоя в работе системных компонентов:

- `Auto` — автоматический режим обновления.
- `Manual` — ручной режим обновления. Для применения обновления нужно установить аннотацию `release.deckhouse.io/disruption-approved=true` на соответствующий ресурс `DeckhouseRelease`.

По умолчанию: `Auto`

Допустимые значения: `Auto`, `Manual`

■ **settings.update.mode**

строка

Режим обновления Deckhouse на выбранном канале обновлений.

- `AutoPatch` — автоматический режим обновления для патч-версий.

Для обновления минорной версии (например, с `v1.69.*` на `v1.70.*`) необходимо подтверждение.

Обновление патч-версии (например, с `v1.70.1` на `v1.70.2`) применяется с учетом окон обновлений, если они заданы.

- **Auto** — автоматический режим обновления для всех версий.

Обновления минорной версии (например, с `v1.69.*` на `v1.70.*`) и патч-версии (например, с `v1.70.1` на `v1.70.2`) применяются с учетом заданных окон обновлений, либо, если окна обновлений не заданы, по мере появления обновлений на соответствующем канале обновлений.

- **Manual** — ручной режим обновления для всех версий.

Для обновления и минорной, и патч-версии необходимо подтверждение.

Для подтверждения обновления версии установите поле `approved` в `true` в соответствующем ресурсе `DeckhouseRelease` необходимо установить поле `approved` в `true`.

Выбранный режим обновления применяется **только** при установленном канале обновлений.

По умолчанию: `AutoPatch`

Допустимые значения: `AutoPatch`, `Auto`, `Manual`

- **settings.update.notification**
объект

Настройка оповещений о запланированном обновлении Deckhouse.

Имеет эффект **только** при установленном автоматическом режиме обновлений.

Пример:

```
notification:
  webhook: https://release-webhook.mydomain.com
  minimalNotificationTime: 8h
  retryMinTime: 2s
```

- **settings.update.notification.auth**
объект

Структура, описывающая способ авторизации на вебхуке.

Если не указано, авторизация не используется.

- **settings.update.notification.auth.basic**
объект

Basic-аутентификация на вебхуке.

- **settings.update.notification.auth.basic.password**
строка

Обязательный параметр

Пароль для авторизации на вебхуке.

Имя и пароль пользователя будут в заголовке `Authorization` в формате `Basic <base64(username:password)>`.

- **settings.update.notification.auth.basic.username**
строка

Обязательный параметр

Имя пользователя для авторизации на вебхуке.

Имя и пароль пользователя будут в заголовке `Authorization` в формате `Basic <base64(username:password)>`.

- **settings.update.notification.auth.bearerToken**
строка

Токен для авторизации на вебхуке.

Токен будет в заголовке `Authorization` в формате `Bearer <token>`.

- **settings.update.notification.minimalNotificationTime**
строка

Минимальное время, которое должно пройти перед обновлением с момента появления новой минорной версии Deckhouse на используемом канале обновлений.

Задается в виде строки с указанием часов и минут: `30m`, `1h`, `2h30m`, `24h`.

Механизм обновления гарантирует, что Deckhouse не обновится, пока не пройдет указанный период времени.

При использовании окон обновлений обновление Deckhouse произойдет в ближайшее возможное окно обновлений, но не ранее чем истечет указанное в `minimalNotificationTime` время.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

Пример:

```
minimalNotificationTime: 6h
```

- **settings.update.notification.releaseType**
строка

Определяет тип версии, о которой будет отправлено оповещение:

- **Minor** — только для обновления минорной версии.
- **All** — для любых обновлений, включая обновление патч-версии.

По умолчанию: `Minor`

Допустимые значения: `All`, `Minor`

Пример:

```
releaseType: All
```

■ **settings.update.notification.retryMinTime** строка

Минимальный интервал между повторными попытками вызова вебхука при неуспешном запросе.

Задается в виде строки с указанием часов, минут или секунд: 30s, 1m, 2m30s, 1h, 2h30m.

Повторные попытки выполняются с экспоненциальной задержкой: каждый следующий интервал в два раза больше предыдущего. Например, при `retryMinTime = 2s` задержки будут: 2s, 4s, 8s, 16s, 32s.

По умолчанию — 2 секунды (если не указано).

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m|[0-9]+s)$`

Пример:

```
retryMinTime: 2s
```

■ **settings.update.notification.tlsSkipVerify** булевый

Пропустить валидацию TLS-сертификата при запросе вебхука.

По умолчанию: `false`

■ **settings.update.notification.webhook** строка

URL-адрес вебхука.

Вызов вебхука произойдет после появления новой минорной версии Deckhouse на используемом канале обновлений, но до момента ее применения в кластере.



Внимание. При указании неверного адреса вебхука, обновление Deckhouse будет заблокировано.



Используйте параметр `minimalNotificationTime` при необходимости установить минимальное время, которое должно пройти перед обновлением с момента появления новой минорной версии Deckhouse на используемом канале обновлений.

На адрес вебхука выполняется POST-запрос с `Content-Type: application/json`. Пример содержания запроса:

```
{
  "subject": "Deckhouse",
  "version": "1.36.0",
  "requirements": {"k8s": "1.20.0"},
  "changelogLink": "https://github.com/deckhouse/deckhouse/changelog/1.36.md",
  "applyTime": "2023-01-01T14:30:00Z00:00",
  "message": "New Deckhouse Release 1.36.0 is available. Release will be applied at: Friday, 01-Jan-23 14:30:00 UTC"
}
```

Описание полей POST-запроса:

- `subject` — строка, тип события обновления. Возможные значения: `Deckhouse`, `Module`;
- `moduleName` — строка, имя модуля. Устанавливается только если `subject: Module`
- `version` — строка в формате `x.y.z` (семантическое версионирование);
- `requirements` — объект, требования к версии;
- `changelogLink` — строка, ссылка на список изменений (changelog) минорной версии;
- `applyTime` — строка, дата и время запланированного обновления (с учетом установленных окон обновлений) в формате `RFC3339`;
- `message` — строка, текстовое сообщение о доступности новой минорной версии и запланированном времени обновления.

Шаблон: `^https?://[^\s/$.?\#].[\s]*$`

Пример:

```
webhook: https://webhook.site/#!/bc8f71ac-c182-4181-9159-6ba6950afffa
```

■ **settings.update.windows** массив объектов

Список окон обновлений в течение суток.

■ **settings.update.windows.days** массив строк

Дни недели, в которые применяется окно обновлений.

Пример:

```
days:
- Mon
- Wed
```

■ **Элемент массива**
строка

День недели.

Допустимые значения: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

Пример:

```
Mon
```

■ **settings.update.windows.from**
строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Должно быть меньше времени окончания окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

■ **settings.update.windows.to**
строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Должно быть больше времени начала окна обновления.

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

ConversionWebhook

Short names: `cwhc`

Scope: Cluster

Version: `v1alpha1`

Кастомный ресурс для настройки вебхуков конвертации версий CRD.

Один кастомный ресурс `ConversionWebhook` описывает обработчики конвертации между версиями пользовательских ресурсов.

- **apiVersion**
строка

`APIVersion` определяет версию схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>.

- **context**
массив объектов

Запускает хук при изменениях объектов Kubernetes.

- **context.kubernetes**
объект

- **context.kubernetes.allowFailure**
булевый

Если `true`, Shell-operator игнорирует ошибки выполнения хука. Если `false` или параметр не установлен, хук перезапускается через 5 секунд после ошибки.

- **context.kubernetes.apiVersion**
строка

Необязательная группа и версия API объекта. Например, `v1` для основных объектов (Pod и т.д.), `rbac.authorization.k8s.io/v1beta1` для `ClusterRole` и `monitoring.coreos.com/v1` для `prometheus-operator`.

- **context.kubernetes.fieldSelector**
объект

- **context.kubernetes.fieldSelector.matchExpressions**
массив объектов

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.field**

строка

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.operator**
строка

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.value**
строка

- **context.kubernetes.includeSnapshotsFrom**
массив строк

Массив имен привязок kubernetes в хуке. При указании, список отслеживаемых объектов из этих привязок будет добавлен в контекст привязки в поле `snapshots`. Возможно само-включение.

- **context.kubernetes.jqFilter**
строка

Необязательный параметр, который указывает фильтрацию событий с использованием синтаксиса jq. Хук будет запущен на событии “Modified” только если результат фильтра изменился после последнего события.

- **context.kubernetes.kind**
строка

Обязательный параметр

Тип отслеживаемого ресурса Kubernetes. Это поле обязательно.

- **context.kubernetes.labelSelector**
объект

Селектор меток для фильтрации ресурсов. Результат `matchLabels` и `matchExpressions` объединяется через `AND`. Пустой селектор меток соответствует всем объектам. Нулевой селектор меток не соответствует ни одному объекту.

- **context.kubernetes.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **context.kubernetes.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **context.kubernetes.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: `In`, `NotIn`, `Exists` и `DoesNotExist`.

- **context.kubernetes.labelSelector.matchExpressions.values**
массив строк

`values` — массив строковых значений. Если в `operator` значение `In` или `NotIn`, массив значений не должен быть пустым. Если в `operator` значение `Exists` или `DoesNotExist`, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **context.kubernetes.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте `matchLabels` эквивалентна элементу `matchExpressions`, ключевое поле которого — `key`, оператор — `In`, а массив значений содержит только «значение». Требования объединяются оператором `AND`.

- **context.kubernetes.nameSelector**
объект

- **context.kubernetes.nameSelector.matchNames**
массив строк

Обязательный параметр

- **context.kubernetes.namespace**
объект

Фильтры для выбора пространств имен.

- **context.kubernetes.namespace.labelSelector**
объект

Селектор меток для фильтрации пространств имен.

- **context.kubernetes.namespace.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **context.kubernetes.namespace.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: `In`, `NotIn`, `Exists` и `DoesNotExist`.

- **context.kubernetes.namespace.labelSelector.matchExpressions.values**
массив строк

`values` — массив строковых значений. Если в `operator` значение `In` или `NotIn`, массив значений не должен быть пустым. Если в `operator` значение `Exists` или `DoesNotExist`, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **context.kubernetes.namespace.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте `matchLabels` эквивалентна элементу `matchExpressions`, ключевое поле которого — `key`, оператор — `In`, а массив значений содержит только «значение». Требования объединяются оператором `AND`.

- **context.kubernetes.namespace.nameSelector**
объект

- **context.kubernetes.namespace.nameSelector.matchNames**
массив строк

Обязательный параметр

- **context.kubernetes.queue**
строка

Имя отдельной очереди. Может использоваться для выполнения долго работающих хуков параллельно с хуками в «основной» очереди.

- **context.name**
строка

Обязательный параметр

Используется для различения разных привязок во время выполнения.

- **conversions**
массив объектов

Обязательный параметр

Эта привязка превращает хук в обработчик для конвертации, определенной в CustomResourceDefinition. Shell-operator обновляет CRD с помощью `.spec.conversion`, запускает HTTPS сервер и выполняет хуки для обработки запросов `ConversionReview`.

- **conversions.from**
строка

Обязательный параметр

Версия пользовательского ресурса, которую хук может конвертировать.

- **conversions.handler**
объект

Обязательный параметр

Код обработчика конвертации.

- **conversions.handler.python**
строка

Python-скрипт обработчика для объекта.

- **conversions.includeSnapshotsFrom**
массив строк

Массив имен привязок `kubernetes` в хуке. При указании, список отслеживаемых объектов из этих привязок будет добавлен в контекст привязки в поле `snapshots`.

- **conversions.to**
строка

Обязательный параметр

Версия пользовательского ресурса, которую хук может производить.

- **kind**
строка

Kind — это строковое значение, представляющее REST-ресурс, который описывает этот объект. Серверы могут определять его на основе конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Записывается в CamelCase. Подробнее: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>.

- **metadata**
объект

- **status**
объект

Status defines the observed state of ConversionWebhook.

ValidationWebhook

Short names: `vwhc`

Scope: Cluster
Version: v1alpha1

Кастомный ресурс для настройки вебхуков валидации Kubernetes ресурсов.

Один кастомный ресурс ValidationWebhook описывает обработчик валидации для определенных типов ресурсов и операций.

- **apiVersion**
строка

APIVersion определяет версию этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>.

- **context**
массив объектов

Запускает хук при изменениях объектов Kubernetes.

- **context.kubernetes**
объект

- **context.kubernetes.allowFailure**
булевый

Если `true`, Shell-operator игнорирует ошибки выполнения хука. Если `false` или параметр не установлен, хук перезапускается через 5 секунд после ошибки.

- **context.kubernetes.apiVersion**
строка

Необязательная группа и версия API объекта. Например, `v1` для основных объектов (Pod и т.д.), `rbac.authorization.k8s.io/v1beta1` для ClusterRole и `monitoring.coreos.com/v1` для prometheus-operator.

- **context.kubernetes.fieldSelector**
объект

- **context.kubernetes.fieldSelector.matchExpressions**
массив объектов

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.field**
строка

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.operator**
строка

Обязательный параметр

- **context.kubernetes.fieldSelector.matchExpressions.value**
строка

- **context.kubernetes.includeSnapshotsFrom**
массив строк

Массив имен привязок kubernetes в хуке. При указании, список отслеживаемых объектов из этих привязок будет добавлен в контекст привязки в поле snapshots. Возможно само-включение.

- **context.kubernetes.jqFilter**
строка

Необязательный параметр, который указывает фильтрацию событий с использованием синтаксиса jq.

Хук будет запущен на событии «Modified» только если результат фильтра изменился после последнего события.

- **context.kubernetes.kind**
строка

Обязательный параметр

Тип отслеживаемого ресурса Kubernetes. Это поле обязательно.

- **context.kubernetes.labelSelector**
объект

Селектор меток для фильтрации ресурсов. Результат `matchLabels` и `matchExpressions` объединяется через `AND`. Пустой селектор меток соответствует всем объектам. Нулевой селектор меток не соответствует ни одному объекту.

- **context.kubernetes.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **context.kubernetes.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **context.kubernetes.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: `In`, `NotIn`, `Exists` и `DoesNotExist`.

- **context.kubernetes.labelSelector.matchExpressions.values**

массив строк

`values` — массив строковых значений. Если в `operator` значение `In` или `NotIn`, массив значений не должен быть пустым. Если в `operator` значение `Exists` или `DoesNotExist`, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **context.kubernetes.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте `matchLabels` эквивалентна элементу `matchExpressions`, ключевое поле которого — `key`, оператор — `In`, а массив значений содержит только «значение». Требования объединяются оператором `AND`.

- **context.kubernetes.nameSelector**
объект

- **context.kubernetes.nameSelector.matchNames**
массив строк

Обязательный параметр

- **context.kubernetes.namespace**
объект

Фильтры для выбора пространств имен.

- **context.kubernetes.namespace.labelSelector**
объект

Селектор меток для фильтрации пространств имен.

- **context.kubernetes.namespace.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция `И`).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **context.kubernetes.namespace.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: `In`, `NotIn`, `Exists` и `DoesNotExist`.

- **context.kubernetes.namespace.labelSelector.matchExpressions.values**
массив строк

`values` — массив строковых значений. Если в `operator` значение `In` или `NotIn`, массив значений не должен быть пустым. Если в `operator` значение `Exists` или `DoesNotExist`, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **context.kubernetes.namespace.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте `matchLabels` эквивалентна элементу `matchExpressions`, ключевое поле которого — `key`, оператор — `In`, а массив значений содержит только «значение». Требования объединяются оператором `AND`.

- **context.kubernetes.namespace.nameSelector**
объект

- **context.kubernetes.namespace.nameSelector.matchNames**
массив строк

Обязательный параметр

- **context.kubernetes.queue**
строка

Имя отдельной очереди. Может использоваться для выполнения долго работающих хуков параллельно с хуками в «основной» очереди.

- **context.name**
строка

Обязательный параметр

Используется для различения разных привязок во время выполнения.

- **handler**
объект

Обязательный параметр

Код обработчика `ValidatingWebhook`.

- **handler.cel**
строка

CEL-обработчик правил для объекта.

- **handler.python**
строка

Python-скрипт обработчика для объекта.

- **kind**
строка

Kind — это строковое значение, представляющее REST-ресурс, который описывает этот объект. Серверы могут определять его на основе конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Записывается в CamelCase. Подробнее: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metadata**
объект
- **status**
объект

Status defines the observed state of ValidationWebhook.

- **validationObject**
объект

Обязательный параметр

ValidatingWebhook описывает вебхук, ресурсы и операции, к которым он применяется.

- **validationObject.failurePolicy**
строка

Определяет, как обрабатываются ошибки от хука.

- **validationObject.group**
строка

Ключ для включения снимков из группы привязок schedule и kubernetes.

- **validationObject.labelSelector**
объект

Селектор меток для фильтрации ресурсов. Результат `matchLabels` и `matchExpressions` объединяется через `AND`. Пустой селектор меток соответствует всем объектам. Нулевой селектор меток не соответствует ни одному объекту.

- **validationObject.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **validationObject.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **validationObject.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: `In`, `NotIn`, `Exists` и `DoesNotExist`.

- **validationObject.labelSelector.matchExpressions.values**
массив строк

`values` — массив строковых значений. Если в `operator` значение `In` или `NotIn`, массив значений не должен быть пустым. Если в `operator` значение `Exists` или `DoesNotExist`, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **validationObject.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте `matchLabels` эквивалентна элементу `matchExpressions`, ключевое поле которого — `key`, оператор — `In`, а массив значений содержит только «значение». Требования объединяются оператором `AND`.

- **validationObject.matchConditions**
массив объектов

Дополнительный список условий соответствия для тонкой фильтрации запросов. Доступно только начиная с версии Kubernetes 1.27.

`matchCondition` представляет собой условие, которое должно быть выполнено для отправки запроса в вебхук.

- **validationObject.matchConditions.expression**
строка

Обязательный параметр

Представляет собой выражение, которое будет оценено CEL. Должно оцениваться как bool. Выражения CEL имеют доступ к содержимому AdmissionRequest и Authorizer, организованному в переменные CEL:

- `object` — объект из входящего запроса. Значение равно `null` для запросов DELETE.
- `oldObject` — существующий объект. Для запросов CREATE значение равно `null`.
- `request` — атрибуты запроса на допуск (/pkg/apis/admission/types.go#AdmissionRequest).
- `authorizer` — авторизатор CEL. Может использоваться для выполнения проверок авторизации для субъекта (пользователя или учетной записи службы) запроса. См. <https://pkg.go.dev/k8s.io/apiserver/pkg/cel/library#Authz>
- `authorizer.requestResource` — CEL ResourceCheck, построенный из `authorizer` и настроенный с помощью ресурса запроса. Документация по CEL: <https://kubernetes.io/docs/reference/using-api/cel/>.

- **validationObject.matchConditions.name**
строка

Обязательный параметр

Является идентификатором для этого условия соответствия, используемым для стратегического объединения MatchConditions, а также для предоставления идентификатора для целей ведения журнала. Хорошее имя должно описывать связанное с ним выражение. Имя должно быть квалифицированным именем, состоящим из алфавитно-цифровых символов, -, , или , и должно начинаться и заканчиваться алфавитно-цифровым символом (например, MyName, или my.name, или 123-abc, регулярное выражение, используемое для проверки, — `(([A-Za-z0-9]+[-A-Za-z0-9]{0,1})?[A-Za-z0-9])` с необязательным префиксом поддомена DNS и / (например, example.com/MyName).

- **validationObject.name**
строка

Обязательный параметр

Должно быть доменом с как минимум тремя сегментами, разделенными точками.

- **validationObject.namespace**
объект

Namespace filtering. Only labelSelector is supported (not nameSelector). To filter by namespace name, use labelSelector with kubernetes.io/metadata.name label.

- **validationObject.namespace.labelSelector**
объект

Селектор меток для фильтрации пространств имен.

- **validationObject.namespace.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений. Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Селектор, содержащий значения, ключ и оператор, который связывает ключ и значения.

- **validationObject.namespace.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Ключ метки, к которому применяется селектор.

- **validationObject.namespace.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Представляет соотношение ключа с набором значений. Допустимые операторы: In, NotIn, Exists и DoesNotExist.

- **validationObject.namespace.labelSelector.matchExpressions.values**
массив строк

values — массив строковых значений. Если в operator значение In или NotIn, массив значений не должен быть пустым. Если в operator значение Exists или DoesNotExist, массив значений должен быть пустым. Этот массив заменяется во время стратегического слияния патчей.

- **validationObject.namespace.labelSelector.matchLabels**
объект

Карта пар {ключ, значение}. Одна пара {ключ, значение} в карте matchLabels эквивалентна элементу matchExpressions, ключевое поле которого — key, оператор — In, а массив значений содержит только «значение». Требования объединяются оператором AND.

- **validationObject.rules**
массив объектов

Обязательный список правил, используемых для определения, следует ли отправить запрос к Kubernetes API server в хук.

RuleWithOperations is a tuple of Operations and Resources. It is recommended to make sure that all the tuple expansions are valid.

- **validationObject.rules.apiGroups**
массив строк

API-группы, к которым принадлежат ресурсы. * означает все группы.

- **validationObject.rules.apiVersions**
массив строк

API-версии, к которым принадлежат ресурсы. * означает все версии.

- **validationObject.rules.operations**
массив строк

Операции, которые интересуют admission hook: CREATE, UPDATE, DELETE, CONNECT или * для всех операций.

- **Элемент массива**
строка

OperationType specifies an operation for a request.

- **validationObject.rules.resources**
массив строк

Список ресурсов, к которым применяется это правило.

- **validationObject.rules.scope**
строка

Определяет область действия этого правила. Допустимые значения: Cluster, Namespaced и *. Cluster означает, что этому правилу будут соответствовать только ресурсы, относящиеся к кластеру. Объекты API пространства имен относятся к кластеру. Namespaced означает, что только ресурсы с пространством имен будут

соответствовать этому правилу. * означает, что нет ограничений по области действия. Подресурсы соответствуют области действия своего родительского ресурса. По умолчанию используется значение *.

- **validationObject.sideEffects**
строка

Определяет, знает ли хук о dryRun.

- **validationObject.timeoutSeconds**
целочисленный

Количество секунд, которое API сервер должен ждать ответа от хука, прежде чем считать вызов неудачным. По умолчанию 10 (секунд).

Примеры конфигурации

Настройка режима обновления

Управлять обновлением DKP можно следующими способами:

- с помощью параметра `settings.update` ресурса `ModuleConfig deckhouse`;
- с помощью секции параметров `disruptions` ресурса `NodeGroup`.

Конфигурация окон обновлений

Управлять временными окнами, когда Deckhouse будет устанавливать обновления автоматически, можно следующими способами:

- для общего управления обновлениями — в параметре `update.windows` ресурса `ModuleConfig deckhouse`;
- для управления обновлениями, которые могут привести к кратковременному простою в работе системных компонентов — в параметрах `disruptions.automatic.windows` и `disruptions.rollingUpdate.windows` ресурса `NodeGroup`.

Пример настройки двух ежедневных окон обновлений: с 8:00 до 10:00 и с 20:00 до 22:00 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: EarlyAccess
    update:
      windows:
        - from: "8:00"
          to: "10:00"
        - from: "20:00"
          to: "22:00"
```

Также можно настроить обновления в определенные дни, например по вторникам и субботам с 18:00 до 19:30 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
      days:
        - Tue
        - Sat
```

Ручное подтверждение обновлений

Ручное подтверждение обновления версии Deckhouse предусмотрено в следующих случаях:

- Включен режим подтверждения обновлений Deckhouse.

Это значит, что параметр `settings.update.mode` `ModuleConfig deckhouse` установлен в `Manual` (подтверждение как patch-версии, так и минорной версии Deckhouse) или в `AutoPatch` (подтверждение минорной версии Deckhouse). Для подтверждения обновления выполните следующую команду (укажите необходимую версию Deckhouse):

```
d8 k patch DeckhouseRelease <VERSION> --type=merge -p='{ "approved": true }'
```

- Если для какой-либо группы узлов отключено автоматическое применение обновлений, которые могут привести к кратковременному простою в работе системных компонентов.

Это значит, что у `NodeGroup`, соответствующего группе узлов, параметр `spec.disruptions.approvalMode` установлен в `Manual`.

Для обновления **каждого** узла в такой группе на узел нужно установить аннотацию `update.node.deckhouse.io/disruption-approved=`. Пример:

```
d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
```

Оповещение об обновлении Deckhouse

В режиме обновлений `Auto` можно настроить вызов вебхука для получения оповещения о предстоящем обновлении минорной версии Deckhouse.

Кроме того, оповещения формируются не только при обновлении Deckhouse, но и при обновлении любых модулей, включая их отдельные обновления. В отдельных случаях система может инициировать отправку нескольких оповещений одновременно (по 10–20 оповещений) с интервалом около 15 секунд.

i Оповещения доступны только в режиме обновлений `Auto`, в режиме `Manual` они не формируются.

i Вебхук указывать не обязательно: если параметр `update.notification.webhook` не задан, но указано время в параметре `update.notification.minimalNotificationTime`, применение новой версии всё равно будет отложено на указанный период. В этом случае оповещением о появлении новой версии можно считать появление в кластере ресурса `DeckhouseRelease` с именем новой версии.

Оповещения отправляются только один раз для конкретного обновления. Если что-то пошло не так (например, вебхук получил некорректные данные), повторная отправка автоматически не произойдёт. Чтобы отправить оповещение повторно, необходимо удалить соответствующий ресурс `DeckhouseRelease`.

Пример настройки оповещения:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
```

После появления новой минорной версии DKP на используемом канале обновлений, но до момента применения ее в кластере на адрес вебхука будет выполнен POST-запрос.

Параметр `minimalNotificationTime` позволяет отложить установку обновления на заданный период, обеспечивая время для реакции на оповещение с учётом окон обновлений. Если при этом вебхук недоступен, каждая неудачная попытка отправки будет сдвигать время применения на ту же величину, что может привести к бесконечному откладыванию обновления.

⚠ Если ваш вебхук возвращает любой код вне диапазона 2xx, DKP повторяет отправку уведомления до пяти раз с экспоненциальной задержкой между попытками. Если все попытки окажутся неуспешными, выпуск блокируется до восстановления доступности вебхука.

Для удобной обработки ошибок и отладки при возврате кодов ошибок вебхук должен возвращать JSON-ответ следующей структуры:

- `code` — необязательный внутренний код ошибки для программной обработки;
- `message` — человекочитаемое описание того, что пошло не так.

Если вебхук возвращает успешный статус HTTP (2xx), DKP считает уведомление успешным вне зависимости от содержимого ответа.

Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h
```

Минимальный пример вебхука на Go...

```

package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
)

// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
    Subject      string      `json:"subject"`
    Version      string      `json:"version"`
    Requirements map[string]string `json:"requirements,omitempty"`
    ChangelogLink string    `json:"changelogLink,omitempty"`
    ApplyTime    string    `json:"applyTime,omitempty"`
    Message      string    `json:"message"`
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
    Code    string `json:"code,omitempty"`
    Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
    }
    defer r.Body.Close()

    var data WebhookData
    if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
        log.Printf("failed to decode payload: %v", err)
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    // Print payload fields
    log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
        data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
    log.Printf("message=%s", data.Message)

    // Example conditional logic: fail intentionally for testing
    if data.Version == "v0.0.0-fail" {
        // Return structured error response with error status code
        errorResp := ResponseError{
            Code:    "TEST_FAILURE",
            Message: "intentional failure for testing",
        }

        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode(errorResp)
        return
    }

    // Return success response with 2xx status code
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Notification processed successfully"))
}

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/webhook", handler)

    addr := ":8080"
    fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
    if err := http.ListenAndServe(addr, mux); err != nil {
        log.Fatal(err)
    }
}

```

Резервирование имён хостов веб-интерфейсов платформы

DKP публикует собственные веб-интерфейсы по именам хостов, которые формируются из глобального параметра `publicDomainTemplate`: API, веб-интерфейс, Grafana, Dex, генератор `kubecfg` и другие. Эти имена зарезервированы в пределах кластера. Ресурсы `Ingress`, `HTTPRoute`, `GRPCRoute`, `TLSRoute`, `ListenerSet` и `Gateway`, созданные вне неймспейса с лейблом `heritage: deckhouse`, занять такое имя не могут — запрос отклоняется с сообщением, в котором указано имя хоста:

```

Hostname console.example.com is reserved for Deckhouse platform services
and cannot be claimed outside a system namespace

```

Проверка выполняется только при создании или изменении ресурсов. Если существующий ресурс уже использует

зарезервированное имя хоста, он продолжает обслуживать трафик до следующего изменения.

Состав резервирования

Состав резервирования задаётся параметром `settings.reservedPublicHosts.mode` ресурса `ModuleConfig` `deckhouse`:

- `Template` (по умолчанию) — резервируются все имена хостов, которые способен сформировать шаблон.
- `List` — резервируются только имена хостов сервисов, о публикации которых DKP знает.

Подробнее о каждом режиме резервирования — в описании параметра.

Исключение из резервирования

В режиме `Template` резервируются не только имена хостов, используемые DKP, но и другие имена, соответствующие шаблону. Если приложению необходимо такое имя, его можно исключить из резервирования одним из следующих способов:

- Освободить одно имя хоста с помощью параметра `settings.reservedPublicHosts.excludedServices`.

В параметре указывается не полное имя хоста, а имя сервиса (подставляемое в шаблон значение). Например, для шаблона `%s.example.com` значение `grafana` исключает из резервирования имя `grafana.example.com`.

- Исключить весь неймспейс с помощью лейбла `security.deckhouse.io/reserved-hosts-bypass: "true"`.

Лейбл устанавливается на неймспейс, а не на объект, поэтому пользователь, которому разрешено управлять ресурсами только в выделенных ему неймспейсах, установить его не сможет.

- Вернуться к прежнему составу резервирования, задав режим `mode: List`.

Пример конфигурации, в котором `grafana.example.com` исключается из резервирования и дополнительно резервируется `admin.corp.example.org`, которое не соответствует шаблону:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    reservedPublicHosts:
      mode: Template
      excludedServices:
        - grafana
    additionalHosts:
      - admin.corp.example.org
```

Параметр `settings.reservedPublicHosts.additionalHosts` позволяет зарезервировать дополнительные имена хостов независимо от значения `excludedServices`.



Параметр `excludedServices` принимает только имя сервиса, поэтому с его помощью нельзя исключить из резервирования wildcard-имя.

Чтобы приложение могло использовать wildcard-имя, например, `*.example.com`, отключите проверку для его неймспейса с помощью упомянутого ранее лейбла или используйте режим `List`.

Исключение для общего Gateway

Gateway, к которому подключаются ресурсы `ListenerSet` модулей DKP, автоматически исключается из резервирования независимо от того, какие имена заявлены в его `listeners`.

Таким Gateway считается:

- объект, указанный в глобальном параметре `gatewayAPIGateway`;
- если параметр не задан — Gateway, указанный в `ConfigMap` `default-gateway` неймспейса `d8-alb`.

Исключение действует по имени и неймспейсу и только для ресурсов Gateway. Оно не распространяется на другие ресурсы, использующие те же имена хостов, например, `Ingress`, `HTTPRoute` или другой Gateway.

Исключение необходимо, потому что такой Gateway обычно содержит `listener` с wildcard-именем и wildcard-сертификатом и зачастую находится в собственном неймспейсе без лейбла `heritage: deckhouse`. Без его исключения было бы невозможно изменить такой Gateway после включения резервирования, хотя от него продолжают зависеть подключённые маршруты DKP.

Любой другой Gateway с `listener *.example.com` отклоняется, и запись о том, что уже обслуживалось, его не покрывает. Чтобы использовать такой Gateway, добавьте его неймспейсу лейбл `security.deckhouse.io/reserved-hosts-bypass: "true"` или включите режим `List`.

Если общий Gateway не указан в параметре `gatewayAPIGateway` и не определён через `ConfigMap` `default-gateway`, автоматическое исключение не применяется.

Резервирование при наличии работающих приложений в кластере

Когда резервирование начинает действовать, имена хостов, уже используемые приложениями, автоматически добавляются в список разрешённых. Без этого их следующее изменение было бы отклонено.

Список хранится в ключе `grandfatheredHosts` объекта `ConfigMap` `d8-reserved-public-hosts` в неймспейсе `d8-system` отдельно от имён, исключённых из резервирования вручную.

Список формируется только при первом включении режима `Template` и впоследствии не обновляется. Это предотвращает обход резервирования: приложение не может занять зарезервированное имя хоста после включения механизма и затем получить разрешение на его использование при повторном формировании списка.

Если приложение, использовавшее разрешённое таким образом имя хоста, удалено и имя больше не требуется, удалите соответствующую запись из `grandfatheredHosts`.

Перед сохранением имени хостов преобразуются в нижний регистр, а точка в конце имени удаляется. Некорректные значения, например URL вместо имени хоста, игнорируются. Если ресурс использует соответствующее зарезервированное имя, запрос на его следующее изменение будет отклонён.

Wildcard-имена (например, *.example.com), в `grandfatheredHosts` не добавляются. Иначе приложение могло бы использовать все имена хостов, соответствующие шаблону, в том числе имена сервисов, которые DKP начнёт публиковать в дальнейшем. Чтобы приложение могло использовать wildcard-имя домена платформы, добавьте его неймспейсу лейбл `security.deckhouse.io/reserved-hosts-bypass: "true"` или включите режим `List`.

Это ограничение не распространяется на Gateway, к которому подключается DKP. Он автоматически исключается из проверки.

Переключение из режима `Template` в `List` и обратно не приводит к повторному формированию `grandfatheredHosts`. Поэтому имена хостов, которые приложения начали использовать в режиме `List`, после возврата в режим `Template` не добавляются в список разрешённых и при следующем изменении соответствующих ресурсов будут проверяться по общим правилам резервирования.

Проверка состояния резервирования

Информация о текущем состоянии резервирования хранится в ConfigMap `d8-reserved-public-hosts` неймспейса `d8-system`.

Чтобы просмотреть ConfigMap, выполните следующую команду:

```
d8 k -n d8-system get configmap d8-reserved-public-hosts -o yaml
```

ConfigMap содержит следующие ключи:

Ключ	Описание
<code>mode</code>	Действующий состав резервирования. Если значение <code>publicDomainTemplate</code> не соответствует схеме, указывается <code>List</code>
<code>hostPattern</code>	Регулярное выражение, описывающее имена хостов, которые способен сформировать шаблон. В режиме <code>List</code> значение будет пустым
<code>hosts</code>	Имена хостов, для которых используется точное сопоставление. В режиме <code>Template</code> содержит значения <code>additionalHosts</code> и wildcard-имя домена, а в режиме <code>List</code> — имена хостов сервисов платформы
<code>allowedHosts</code>	Имена хостов, исключённые из проверки по <code>hostPattern</code> : имена из <code>excludedServices</code> и <code>grandfatheredHosts</code>
<code>excludedHosts</code>	Имена хостов, исключённые из резервирования через <code>excludedServices</code>
<code>grandfatheredHosts</code>	Имена хостов, которые использовались приложениями при включении резервирования и были автоматически добавлены в список разрешённых
<code>sharedGateway</code>	Gateway, к которому подключается DKP, в формате <code><NAMESPACE>/<NAME></code> . Если такой Gateway не задан и не обнаружен, значение будет пустым
<code>platformHosts</code>	Имена хостов сервисов, публикуемых DKP. Используется как справочный ключ для обоих режимов
<code>unknownExcludedServices</code>	Имена из <code>excludedServices</code> , которые не публикует ни один модуль. Наличие имени скорее всего указывает на опечатку в имени сервиса

Сбор информации для отладки

О сборе отладочной информации читайте в FAQ.

FAQ

Как запустить kube-bench в кластере?

- 1. Зайдите внутрь пода Deckhouse:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- bash
```

- 2. Выберите, на каком узле запустить kube-bench.

- Запуск на случайном узле:

```
curl -S https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k create -f -
```

- Запуск на конкретном узле, например на control-plane:

```
curl -S https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k apply -f - --dry-run=client -o json | jq '.spec.template.spec.tolerations=[{"operator": "Exists"}] | .spec.template.spec.nodeSelector={"node-role.kubernetes.io/control-plane": ""}' | d8 k create -f -
```

- 3. Проверьте результат выполнения:

```
d8 k logs job.batch/kube-bench
```



В Deckhouse установлен срок хранения логов — 7 дней. Однако, в соответствии с требованиями безопасности указанными в kube-bench, логи должны храниться не менее 30 дней. Используйте отдельное хранилище для логов, если вам необходимо хранить логи более 7 дней.

Как собрать информацию для отладки?

1. Соберите диагностический архив утилитой `d8`, перенаправив вывод (stdout) в файл:

```
d8 system collect-debug-info > deckhouse-debug-$(date +%Y_%m_%d").tar.gz
```

2. Отправьте полученный архив команде Deckhouse для дальнейшего расследования.



Флаг `--exclude` исключает указанные элементы из архива. Пример:

```
d8 system collect-debug-info --exclude=queue global-values > deckhouse-debug-$(date +%Y_%m_%d").tar.gz
```



Флаг `--list-exclude` выводит список доступных для исключения элементов. Пример:

```
d8 system collect-debug-info --list-exclude
```

Ниже перечислены сведения, формируемые при выполнении архива. Названия в колонке «Файл в архиве» соответствуют элементам верхнего уровня внутри итогового архива `tar.gz`. Отдельные чувствительные значения (например, `kubeRBACProxyCA` и `registry.dockercfg`) исключаются из выборки.

Категория	Собираемые данные	Файл в архиве
Deckhouse	Состояние очереди Deckhouse	queue
	Значения Deckhouse (кроме <code>kubeRBACProxyCA</code> и <code>registry.dockercfg</code>)	global-values
	Версия текущего пода <code>deckhouse</code>	deckhouse-version
	Все объекты <code>DeckhouseRelease</code>	deckhouse-releases
	Логи подов Deckhouse	deckhouse-logs
	Манифесты контроллеров и подов из всех пространств имён Deckhouse	d8-all
Объекты кластера	<code>NodeGroup</code>	node-groups
	<code>NodeGroupConfiguration</code>	node-group-configuration
	<code>Node</code>	nodes
	<code>Machine</code>	machines
	<code>Instance</code>	instances
	<code>StaticInstance</code>	staticinstances
	<code>MachineDeployment</code>	cloud-machine-deployment, static-machine-deployment
	<code>ClusterAuthorizationRule</code>	cluster-authorization-rules
	<code>AuthorizationRule</code>	authorization-rules
	<code>ModuleConfig</code>	module-configs
	События (все пространства имён)	events
Модули и их состояния	Список включённых модулей	deckhouse-enabled-modules
	Объекты <code>ModuleSource</code> в кластере	deckhouse-module-sources
	Объекты <code>ModulePullOverride</code> в кластере	deckhouse-module-pull-overrides
	Модули в режиме <code>maintenance</code>	deckhouse-maintenance-modules
Логи и манифесты контроллеров	Логи <code>machine-controller-manager</code>	mcm-logs
	Логи <code>cloud-controller-manager</code>	ccm-logs
	Логи <code>csi-controller</code>	csi-controller-logs
	Логи <code>cluster-autoscaler</code>	cluster-autoscaler-logs
	Логи <code>VPA admission controller</code>	vpa-admission-controller-logs
	Логи <code>VPA recommender</code>	vpa-recommender-logs
	Логи <code>VPA updater</code>	vpa-updater-logs
	YAML <code>capi-controller-manager</code>	capi-controller-manager
	YAML <code>caps-controller-manager</code>	caps-controller-manager
	YAML <code>machine-controller-manager</code>	machine-controller-manager
Мониторинг и алерты	Логи Prometheus	prometheus-logs

Категория	Собираемые данные	Файл в архиве
	Активные (горящие) уведомления в Prometheus	alerts
	Поды не в состоянии Running (кроме Completed и Evicted)	bad-pods
	Список Audit Policy	audit-policy
Сеть	Все объекты в пространстве имён d8-istio	d8-istio-resources
	Все кастомные ресурсы istio	d8-istio-custom-resources
	Конфигурация Envoy для istio	d8-istio-envoy-config
	Логи istio	d8-istio-system-logs
	Логи istio ingress gateway	d8-istio-ingress-logs
	Логи istio users	d8-istio-users-logs
	Состояние соединения Cilium (cilium health status)	cilium-health-status

Как отлаживать проблемы в подах с помощью ephemeral containers?

Выполните следующую команду:

```
d8 k -n <namespace_name> debug -it <pod_name> --image=ubuntu <container_name>
```

Подробнее можно почитать в официальной документации.

Как отлаживать проблемы на узлах с помощью ephemeral containers?

Выполните следующую команду:

```
d8 k debug node/mynode -it --image=ubuntu
```

Подробнее можно почитать в официальной документации.

Модуль deckhouse-tools

i

Веб-интерфейсы, связанные с модулем: deckhouse-tools

Этот модуль создает веб-интерфейс со ссылками на скачивание утилиты Deckhouse CLI под различные операционные системы.

Адрес веб-интерфейса формируется в соответствии с шаблоном publicDomainTemplate глобального параметра конфигурации Deckhouse (ключ %s заменяется на tools).

Например, если publicDomainTemplate установлен как %s-kube.company.my, веб-интерфейс будет доступен по адресу tools-kube.company.my.

Настройки

У модуля нет обязательных настроек.

Пример конфигурации

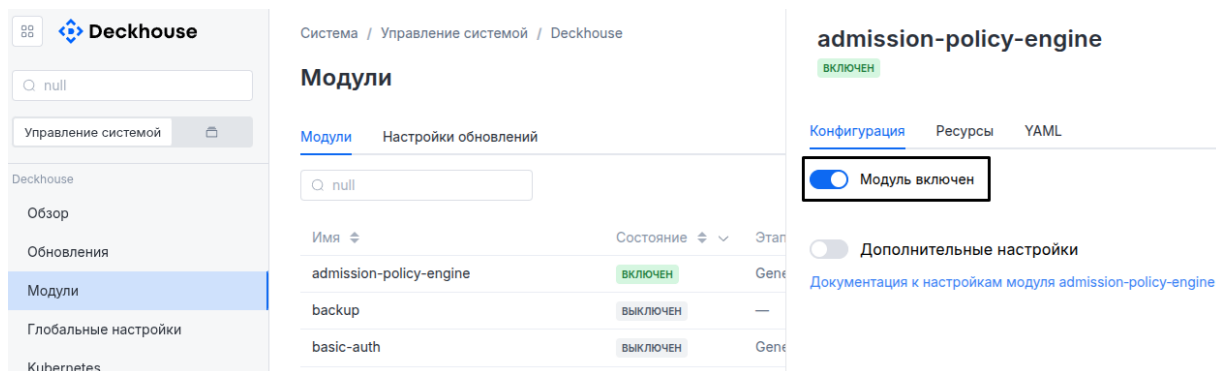
```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  enabled: true
  version: 1
```

Модуль **включен** по умолчанию в наборах модулей: Default, Managed. Модуль **выключен** по умолчанию в наборе модулей Minimal.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль deckhouse-tools, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `deckhouse-tools`:

```
d8 system module enable deckhouse-tools
```

- **С помощью ModuleConfig `deckhouse-tools`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `deckhouse-tools` (создайте его, при необходимости).

Пример манифеста для включения модуля `deckhouse-tools`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  enabled: true
```

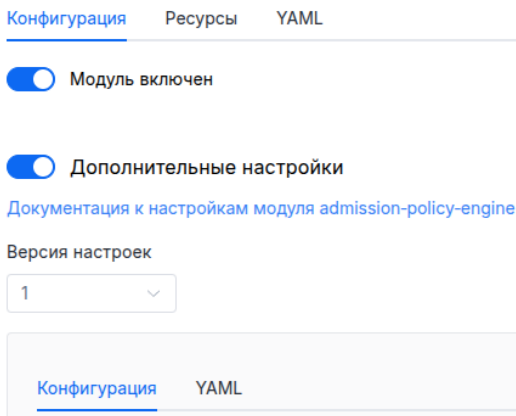
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `deckhouse-tools`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `deckhouse-tools` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `deckhouse-tools`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `deckhouse-tools` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc deckhouse-tools
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `deckhouse-tools`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```


Пример файла манифеста ModuleConfig `deckhouse-tools`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.auth**

объект

Параметры для аутентификации и авторизации доступа к веб-интерфейсу модуля.

- **settings.auth.allowedUserEmails**

массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.allowedUserGroups**

массив строк

Массив из групп, пользователи которых имеют доступ к веб-интерфейсу модуля.

Этот параметр используется, если включен модуль `user-authn` или установлен параметр `externalAuthentication`.



Внимание. Эти группы необходимо добавить в соответствующее поле в конфигурации `DexProvider`, если этот модуль используется вместе с модулем `user-authn`.

По умолчанию: `[]`

Примеры:

```
allowedUserGroups:
- admin
- users
```

```
allowedUserGroups: []
```

- **settings.auth.externalAuthentication**

объект

Параметры для включения внешней аутентификации на основе Ingress NGINX `external-auth` механизма, использующего Nginx `auth_request` module.



Внешняя аутентификация включена по умолчанию, если включен модуль `user-authn`.

- **settings.auth.externalAuthentication.authSignInURL**

строка

URL-адрес для перенаправления пользователя на проверку аутентификации (если служба аутентификации вернула код ответа HTTP, отличный от 200).

По умолчанию:

Пример:

```
authSignInURL: https://$host/dex-authenticator/sign_in
```

- **settings.auth.externalAuthentication.authURL**

строка

URL-адрес сервиса аутентификации.

Если пользователь прошел аутентификацию, сервис должен вернуть код ответа HTTP 200.

По умолчанию:

Пример:

```
authURL: https://deckhouse-tools-dex-authenticator.d8-system.svc.cluster.local/dex-authenticator/auth
```

- **settings.https**

объект

Тип используемого сертификата.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.https.certManager** объект

Настройки для cert-manager.

■ **settings.https.certManager.clusterIssuerName** строка

Тип ClusterIssuer'a, используемого для заказа SSL-сертификата (доступны `letsencrypt`, `letsencrypt-staging` и `selfsigned`, но возможно определить свои).

По умолчанию: `letsencrypt`

Пример:

```
clusterIssuerName: letsencrypt
```

■ **settings.https.customCertificate** объект

Настройки для использования пользовательского сертификата.

■ **settings.https.customCertificate.secretName** строка

Имя Secret'a в пространстве имен `d8-system`, который будет использоваться для веб-интерфейса модуля.

Secret должен быть в формате `kubernetes.io/tls`.

■ **settings.https.mode** строка

Режим работы HTTPS:

- `CertManager` — доступ по HTTPS с заказом сертификата согласно ClusterIssuer'у, заданному в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — доступ по HTTPS с использованием сертификата из пространства имен `d8-system`.
- `Disabled` — доступ только по HTTP.
- `OnlyInURI` — доступ по HTTP, подразумевая, что перед веб-интерфейсом стоит внешний HTTPS-балансер, который терминирует HTTPS, и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `CertManager`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

○ **settings.ingressClass** строка

Класс Ingress-контроллера веб-интерфейса модуля.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

○ **settings.nodeSelector** объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, `nodeSelector` будет определяться автоматически.

Пример:

```
nodeSelector:
  disktype: ssd
```

○ **settings.tolerations** массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, `tolerations` будет определяться автоматически.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Модуль descheduler

Каждые 15 минут модуль анализирует состояние кластера и выполняет вытеснение подов, соответствующих условиям, описанным в активных стратегиях. Вытесненные поды вновь проходят процесс планирования с учетом текущего состояния кластера. Это позволяет перераспределить рабочие нагрузки в соответствие с выбранной стратегией.

Модуль основан на проекте descheduler.

Особенности работы модуля

- Модуль может учитывать класс приоритета пода (параметр `spec.priorityClassThreshold`), ограничивая работу только подами, у которых класс приоритета ниже заданного порога;
- Модуль не вытесняет под в следующих случаях:
 - под находится в пространстве имен `d8-*` или `kube-system`;
 - под имеет `priorityClassName` `system-cluster-critical` или `system-node-critical`;
 - под связан с локальным хранилищем;
 - под связан с `DaemonSet`;
 - вытеснение пода нарушит `Pod Disruption Budget (PDB)`;
 - нет доступных узлов для запуска вытесненного пода.
- Поды с классом приоритета `Best effort` вытесняются раньше, чем поды с классами `Burstable` и `Guaranteed`.

Для фильтрации подов и узлов модуль использует механизм `labelSelector` Kubernetes:

- `podLabelSelector` — ограничивает поды по лейблам;
- `namespaceLabelSelector` — фильтрует поды по пространствам имен.
- `nodeLabelSelector` — выбирает узлы по лейблам.

Поставщик метрик

Если в кластере зарегистрирована API-группа `metrics.k8s.io` (например, установлен `metrics-server`), модуль автоматически обнаруживает её и включает поставщик метрик **KubernetesMetrics** в политике `descheduler`. Это позволяет при установленной стратегии утилизации `LowNodeUtilization` использовать данные о **фактическом** потреблении ресурсов из Metrics API, а не полагаться исключительно на запросы и лимиты подов. Соответственно, значения в параметрах `thresholds` и `targetThresholds` начинают трактоваться как процент от фактического потребления, а не от запросов подов. На стратегию `HighNodeUtilization` это не влияет: она всегда использует запросы и лимиты подов.

От пользователя не требуется никаких действий: модуль обнаруживает API-группу `metrics.k8s.io`, отслеживая ресурсы `APIService`, и настраивает политику автоматически. Если `metrics-server` установлен **после** запуска модуля, под `descheduler` будет автоматически перезапущен с обновлённой политикой.

i Если `metrics.k8s.io` недоступен, модуль использует поведение по умолчанию: потребление ресурсов оценивается на основе запросов и лимитов подов.

Вытеснение в фоновом режиме

Модуль запускает `descheduler` с включённым feature gate `EvictionsInBackground` из upstream-проекта (`--feature-gates EvictionsInBackground=true`). Отдельной настройки модуля для него нет: поведение заложено в модуль и не требует конфигурации.

По умолчанию `descheduler` рассчитывает, что каждое вытеснение завершается сразу: запрос к `eviction API` либо выполняется, либо отклоняется. Некоторые нагрузки так вытеснить нельзя. Например, поду, обслуживающему виртуальную машину, сначала требуется живая миграция, которая может занимать минуты и требовать повторных попыток.

Если feature gate включён, вытеснение пода с аннотацией `descheduler.alpha.kubernetes.io/request-evict-only` считается *запросом* на запуск вытеснения. Контроллер, которому принадлежит под, отклоняет запрос к `eviction API` с кодом `429 Too Many Requests` и проставляет поду аннотацию `descheduler.alpha.kubernetes.io/eviction-in-progress` на время фактического вытеснения (например, живой миграции). `Descheduler` отслеживает такие поды и учитывает их в своих лимитах вытеснения, пока под не будет удалён или пока с него не снимут аннотацию. Благодаря этому он не запускает больше вытеснений, чем разрешено, пока предыдущие ещё выполняются.

Поды без аннотации `descheduler.alpha.kubernetes.io/request-evict-only` вытесняются как обычно, поэтому в кластерах, где нет нагрузок с собственной политикой вытеснения, ничего не меняется.

i Feature gate имеет статус alpha в upstream-проекте `descheduler` (см. KEP-1397). Сам по себе он ничего не меняет: без контроллера, отвечающего на запрос вытеснения описанным образом, `descheduler` работает ровно так же, как раньше.

Стратегии

HighNodeUtilization

- i** Концентрирует нагрузку на меньшем числе узлов. Требует настройку планировщика и включение автоматического масштабирования.
Чтобы использовать `HighNodeUtilization`, необходимо явно указать профиль планировщика `high-node-utilization` для каждого пода (этот профиль не может быть установлен как профиль по умолчанию).

Стратегия определяет *недостаточно нагруженные узлы* и вытесняет с них поды, чтобы распределить их компактнее, на меньшем числе узлов.

Недостаточно нагруженный узел — узел, использование ресурсов которого меньше **всех** пороговых значений, заданных в секции параметров `strategies.highNodeUtilization.thresholds`.

Стратегия включается параметром `spec.strategies.highNodeUtilization.enabled`.

- ⚠** В GKE нельзя настроить конфигурацию планировщика по умолчанию, но можно использовать стратегию `optimize-utilization` или развернуть второй пользовательский планировщик.

- ⚠** Использование ресурсов узла учитывает `extended-ресурсы` и рассчитывается на основе запросов и лимитов подов (requests and limits), а не их фактического потребления. Такой подход обеспечивает согласованность с работой kube-scheduler, который использует аналогичный принцип при размещении подов на узлах.

Стратегия всегда использует запросы и лимиты подов, независимо от того, доступна ли в кластере API-группа `metrics.k8s.io` (см. Поставщик метрик).

LowNodeUtilization

- i** Более равномерно нагружает узлы.

Стратегия выявляет *недостаточно нагруженные узлы* и вытесняет поды с других, *избыточно нагруженных узлов*. Стратегия предполагает, что пересоздание вытесненных подов произойдет на недостаточно нагруженных узлах (при обычном поведении планировщика).

Недостаточно нагруженный узел — узел, использование ресурсов которого меньше **всех** пороговых значений, заданных в секции параметров `strategies.lowNodeUtilization.thresholds`.

Избыточно нагруженный узел — узел, использование ресурсов которого больше **хотя бы одного** из пороговых значений, заданных в секции параметров `strategies.lowNodeUtilization.targetThresholds`.

Узлы с использованием ресурсов в диапазоне между `thresholds` и `targetThresholds` считаются оптимально используемыми. Поды на таких узлах вытесняться не будут.

Стратегия включается параметром `spec.strategies.lowNodeUtilization.enabled`.

- ⚠** Если API-группа `metrics.k8s.io` в кластере **недоступна**, использование ресурсов узла учитывает `extended-ресурсы` и рассчитывается на основе запросов и лимитов подов (requests and limits), а не их фактического потребления. Такой подход обеспечивает согласованность с работой kube-scheduler, который использует аналогичный принцип при размещении подов на узлах.

Если API-группа `metrics.k8s.io` **доступна** (см. Поставщик метрик), стратегия использует данные о **фактическом** потреблении ресурсов из Metrics API *вместо* запросов и лимитов. В этом случае значения, которыми оперирует descheduler, близки к выводу `kubectl top`, но перестают совпадать с оценкой kube-scheduler: узел с исчерпанными запросами может считаться недостаточно нагруженным, и поды с него вытесняться не будут.

RemoveDuplicates

- i** Предотвращает запуск нескольких подов одного контроллера (ReplicaSet, ReplicationController, StatefulSet) или заданий (Job) на одном узле.

Стратегия следит за тем, чтобы на одном узле не находилось больше одного пода ReplicaSet, ReplicationController, StatefulSet или подов одного задания (Job). Если таких подов два или больше, модуль вытесняет лишние поды, чтобы они лучше распределились по кластеру.

Описанная ситуация может возникнуть, если некоторые узлы кластеры вышли из строя по каким-либо причинам, и поды с них были перемещены на другие узлы. Как только вышедшие из строя узлы снова станут доступны для приема нагрузки, эту стратегию можно будет использовать для выселения дублирующих подов с других узлов.

Стратегия включается параметром `strategies.removeDuplicates.enabled`.

RemovePodsHavingTooManyRestarts

Стратегия вытесняет с узлов поды со слишком большим количеством перезапусков. Кандидатами на вытеснение становятся поды, у которых суммарное число перезапусков всех контейнеров, включая `init-контейнеры`, превышает порог `podRestartThreshold`. Полезна для вытеснения подов в состоянии `CrashLoopBackOff` или с постоянными сбоями, а также для освобождения ресурсов и возможности планирования новых подов на потенциально более здоровых узлах.

RemovePodsViolatingInterPodAntiAffinity

- i** Вытесняет поды, нарушающие правила inter-pod affinity и anti-affinity.

Стратегия гарантирует, что поды, нарушающие правила inter-pod affinity и anti-affinity, будут удалены с узлов.

Например, если на узле находится **Под1**, а также **Под2** и **Под3**, имеющие правила anti-affinity, которые запрещают им работать на одном узле с подом **Под1**, то **Под1** будет вытеснен с узла, чтобы **Под2** и **Под3** смогли работать. Такая ситуация может возникнуть, когда правила inter-pod affinity для **Под2** и **Под3** создаются когда поды уже запущены на узле.

Стратегия включается параметром `strategies.removePodsViolatingInterPodAntiAffinity.enabled`.

RemovePodsViolatingNodeAffinity

i Вытесняет поды, нарушающие правила node affinity.

Стратегия гарантирует, что все поды, которые нарушают правила node affinity, в конечном счете будут удалены с узлов.

По сути, в зависимости от настроек параметра `strategies.removePodsViolatingNodeAffinity.nodeAffinityType`, стратегия превращает правило `requiredDuringSchedulingIgnoredDuringExecution` node affinity пода в правило `requiredDuringSchedulingRequiredDuringExecution`, а правило `preferredDuringSchedulingIgnoredDuringExecution` в правило `preferredDuringSchedulingPreferredDuringExecution`.

Пример для `nodeAffinityType: requiredDuringSchedulingIgnoredDuringExecution`. Есть под, который был назначен на узел, соответствующий правилу `requiredDuringSchedulingIgnoredDuringExecution` node affinity на момент размещения. Если со временем этот узел перестанет удовлетворять правилу node affinity, и если появится другой доступный узел, соответствующий этому правилу, стратегия вытеснит под с узла, на который он был изначально назначен.

Пример для `nodeAffinityType: preferredDuringSchedulingIgnoredDuringExecution`. Есть под, который был назначен на узел, т.к. на момент размещения отсутствовали другие узлы, удовлетворяющие правилу `preferredDuringSchedulingIgnoredDuringExecution` node affinity. Если со временем в кластере появится доступный узел, соответствующий этому правилу, стратегия вытеснит под с узла, на который он был изначально назначен.

Стратегия включается параметром `strategies.removePodsViolatingNodeAffinity.enabled`.

RemovePodsViolatingTopologySpreadConstraint

Стратегия гарантирует, что поды, нарушающие ограничения распределения по топологии, будут вытеснены с узлов. Вытесняется минимальное количество подов, необходимое для приведения доменов топологии в соответствие с `maxSkew` каждого ограничения. Стратегия полезна для перебалансировки подов между зонами доступности после восстановления зоны из аварии.

Настройки

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `descheduler`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left, there's a sidebar with navigation links: Обзор, Обновления, Модули (selected), Глобальные настройки, Kubernetes. The main area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. Below this, there's a table of modules:

Имя	Состояние	Этап
admission-policy-engine	ВКЛЮЧЕН	Генерация
backup	ВЫКЛЮЧЕН	—
basic-auth	ВЫКЛЮЧЕН	Генерация

On the right, the 'admission-policy-engine' module configuration is shown. It has a toggle switch for 'Модуль включен' which is turned on. There are also tabs for 'Конфигурация', 'Ресурсы', and 'YAML'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `descheduler`:

```
d8 system module enable descheduler
```

- **С помощью ModuleConfig `descheduler`.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig descheduler` (создайте его, при необходимости).

Пример манифеста для включения модуля `descheduler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `descheduler`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `descheduler` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `descheduler`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `descheduler` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc descheduler
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `descheduler`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `descheduler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Удалите `settings`.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.deschedulingInterval**
строка

Интервал между запусками цикла перераспределения подов.

Поддерживаются следующие значения:

- `Frequent` — запуск каждые 5 минут. Подходит для кластеров, где важно быстрее перераспределять поды;
- `Moderate` — запуск каждые 15 минут (по умолчанию);
- `Rare` — запуск каждые 30 минут. Подходит для кластеров, где важно минимизировать количество перераспределений подов.

По умолчанию: `Moderate`

Допустимые значения: `Frequent`, `Moderate`, `Rare`

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Descheduler

Scope: Cluster

- v1alpha2
- v1alpha1

Описание одного экземпляра descheduler

- **apiVersion**
строка
- **kind**
строка
- **metadata**
объект
 - **metadata.name**
строка
- **spec**
объект

Обязательный параметр

Описывает поведение экземпляра descheduler.

- **spec.evictLocalStoragePods**
булевый

Позволяет вытеснять поды, использующие локальное хранилище.

По умолчанию: `false`

- **spec.namespaceLabelSelector**
объект

Ограничение обработки подов с помощью лейблов пространства имен.

- **spec.namespaceLabelSelector.matchExpressions**
массив объектов

Список выражений лейблов, которые должен иметь узел, чтобы удовлетворить условию фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.namespaceLabelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.namespaceLabelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.namespaceLabelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: `1..63`

- **spec.namespaceLabelSelector.matchLabels**
объект

- **spec.nodeLabelSelector**
объект

Ограничение обрабатываемых подов для соответствия вытесненным подам по лейблов в представлении множества (set representation). Если значение установлено, не используйте `nodeSelector`.

- **spec.nodeLabelSelector.matchExpressions**
массив объектов

Список выражений лейблов, которые должен иметь узел, чтобы удовлетворить условию фильтра.

Пример:

```
matchExpressions:
```

- key: tier
 - operator: In
 - values:
 - production
 - staging
- key: tier
 - operator: NotIn
 - values:
 - production

- **spec.nodeLabelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.nodeLabelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.nodeLabelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: 1..63

- **spec.nodeLabelSelector.matchLabels**
объект

- **spec.podLabelSelector**
объект

Ограничение обработки подов по лейблам.

- **spec.podLabelSelector.matchExpressions**
массив объектов

Список выражений лейблов, которые должен иметь узел, чтобы удовлетворить условию фильтра.

Пример:

```
matchExpressions:
```

- key: tier
 - operator: In
 - values:
 - production
 - staging
- key: tier
 - operator: NotIn
 - values:
 - production

- **spec.podLabelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.podLabelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.podLabelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: 1..63

- **spec.podLabelSelector.matchLabels**
объект

- **spec.priorityClassThreshold**
объект

Ограничение обработки подов по классу приоритета. Только поды, чей приоритет ниже установленного значения, могут быть вытеснены.

Можно указать **либо** название класса приоритета (`priorityClassThreshold.name`), **либо** непосредственно значение класса приоритета (`priorityThreshold.value`).

По умолчанию это значение установлено на класс приоритета `system-cluster-critical`.

- **spec.priorityClassThreshold.name**
строка

Имя класса приоритета.

- **spec.priorityClassThreshold.value**
целочисленный

Значение класса приоритета.

- **spec.strategies**
объект

Обязательный параметр

Настройки стратегий данного экземпляра ресурса Descheduler.

- **spec.strategies.highNodeUtilization**
объект

Стратегия определяет недостаточно нагруженные узлы и вытесняет с них поды, чтобы распределить их компактнее, на меньшем числе узлов.

Недостаточно нагруженный узел — узел, использование ресурсов которого меньше **всех** пороговых значений, заданных в секции параметров thresholds.

Стратегия предназначена для использования совместно с авто-масштабированием, чтобы сокращать количество недостаточно нагруженных узлов. При использовании стратегии необходимо также настроить стратегию оценки планировщика как `MostAllocated`.

i

В GKE нельзя настроить конфигурацию планировщика по умолчанию, но можно использовать стратегию `optimize-utilization` или развернуть второй пользовательский планировщик.

Использование ресурсов узла учитывает extended-ресурсы и рассчитывается на основе запросов и лимитов подов, а не фактическом потреблении.

- **spec.strategies.highNodeUtilization.enabled**
булевый

Делает стратегию активной.

По умолчанию: `false`

- **spec.strategies.highNodeUtilization.thresholds**
объект

Устанавливает пороговые значения для определения недостаточно нагруженных узлов.

Если использование ресурсов узла меньше **всех** пороговых значений, то узел считается недостаточно нагруженным.

Если порог для ресурса не задан, считается, что он равен 100%, и данный ресурс не участвует в определении недостаточно или избыточно нагруженных узлов.

- **spec.strategies.highNodeUtilization.thresholds.cpu**
целочисленный

Процент отношения запрошенных ресурсов по CPU на узле к доступным.

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.highNodeUtilization.thresholds.memory**
целочисленный

Процент отношения запрошенных ресурсов по памяти на узле к доступным.

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.highNodeUtilization.thresholds.pods**
целочисленный

Процент количества подов от установленной емкости узла.

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization**
объект

Стратегия выявляет *недостаточно нагруженные узлы* и вытесняет поды с других, *избыточно нагруженных узлов*. Стратегия предполагает, что пересоздание вытесненных подов произойдет на недостаточно нагруженных узлах (при обычном поведении планировщика).

Недостаточно нагруженный узел — узел, использование ресурсов которого меньше **всех** пороговых значений, заданных в секции параметров thresholds.

Избыточно нагруженный узел — узел, использование ресурсов которого больше **хотя бы одного** из пороговых значений, заданных в секции параметров targetThresholds.

Использование ресурсов узла учитывает extended-ресурсы и рассчитывается на основе запросов и лимитов подов, а не фактическом потреблении.

Если в кластере зарегистрирована API-группа `metrics.k8s.io`, модуль автоматически переключает эту стратегию на **фактическое** потребление ресурсов по данным Metrics API.

- **spec.strategies.lowNodeUtilization.enabled**
булевый

Делает стратегию активной.

По умолчанию: `false`

- **spec.strategies.lowNodeUtilization.targetThresholds**

объект

Устанавливает пороговые значения для определения избыточно нагруженных узлов.

Если использование ресурсов узла больше **хотя бы одного** из пороговых значений, то узел считается избыточно нагруженным.

Если порог для ресурса не задан, считается, что он равен 100%, и данный ресурс не участвует в определении недостаточно или избыточно нагруженных узлов.

- **spec.strategies.lowNodeUtilization.targetThresholds.cpu**

целочисленный

Процент отношения используемых ресурсов CPU на узле к доступным (по запросам подов либо, если в кластере зарегистрирована API-группа `metrics.k8s.io`, по фактическому потреблению).

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.memory**

целочисленный

Процент отношения используемых ресурсов памяти на узле к доступным (по запросам подов либо, если в кластере зарегистрирована API-группа `metrics.k8s.io`, по фактическому потреблению).

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.pods**

целочисленный

Процент количества подов от установленной емкости узла.

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.thresholds**

объект

Устанавливает пороговые значения для определения недостаточно нагруженных узлов.

Если использование ресурсов узла меньше **всех** пороговых значений, то узел считается недостаточно нагруженным.

Если порог для ресурса не задан, считается, что он равен 100%, и данный ресурс не участвует в определении недостаточно или избыточно нагруженных узлов.

- **spec.strategies.lowNodeUtilization.thresholds.cpu**

целочисленный

Процент отношения используемых ресурсов CPU на узле к доступным (по запросам подов либо, если в кластере зарегистрирована API-группа `metrics.k8s.io`, по фактическому потреблению).

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.thresholds.memory**

целочисленный

Процент отношения используемых ресурсов памяти на узле к доступным (по запросам подов либо, если в кластере зарегистрирована API-группа `metrics.k8s.io`, по фактическому потреблению).

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.thresholds.pods**

целочисленный

Процент количества подов от установленной емкости узла.

Допустимые значения: `0 <= X <= 100`

- **spec.strategies.removeDuplicates**

объект

Стратегия следит за тем, чтобы на одном узле не находилось больше одного пода ReplicaSet, ReplicationController, StatefulSet или подов одного задания (Job). Если таких подов два или больше, модуль вытесняет лишние поды, чтобы они лучше распределились по кластеру.

- **spec.strategies.removeDuplicates.enabled**

булевый

Делает стратегию активной.

По умолчанию: `false`

- **spec.strategies.removePodsHavingTooManyRestarts**

объект

Стратегия вытесняет с узлов поды со слишком большим количеством перезапусков.

Кандидатами на вытеснение становятся поды, у которых суммарное число перезапусков всех контейнеров, включая init-контейнеры, превышает порог `podRestartThreshold`.

Стратегия полезна для вытеснения подов в состоянии `CrashLoopBackOff` или с постоянными сбоями, а также для освобождения ресурсов и возможности планирования новых подов на потенциально более здоровых узлах.

- **spec.strategies.removePodsHavingTooManyRestarts.enabled**

булевый

Делает стратегию активной.

По умолчанию: `false`

- **spec.strategies.removePodsHavingTooManyRestarts.podRestartThreshold**

целочисленный

Порог суммарного количества перезапусков всех контейнеров пода, включая init-контейнеры.

Если число перезапусков превышает это значение, под считается проблемным и становится кандидатом на вытеснение.

По умолчанию: `100`

Допустимые значения: `1 <= X`

■ **spec.strategies.removePodsViolatingInterPodAntiAffinity** объект

Стратегия гарантирует, что поды, нарушающие правила inter-pod affinity и anti-affinity, будут удалены с узлов.

Такая ситуация может возникнуть, если правила антипривязки для подов `podB` и `podC` создаются после того, как они уже запущены на узле.

■ **spec.strategies.removePodsViolatingInterPodAntiAffinity.enabled** булевый

Делает стратегию активной.

По умолчанию: `false`

■ **spec.strategies.removePodsViolatingNodeAffinity** объект

Стратегия гарантирует, что все поды, которые нарушают правила node affinity, в конечном счете будут удалены с узлов.

По сути, в зависимости от настроек параметра `nodeAffinityType`, стратегия превращает правило `requiredDuringSchedulingIgnoredDuringExecution` node affinity пода в правило `requiredDuringSchedulingRequiredDuringExecution`, а правило `preferredDuringSchedulingIgnoredDuringExecution` в правило `preferredDuringSchedulingPreferredDuringExecution`.

■ **spec.strategies.removePodsViolatingNodeAffinity.enabled** булевый

Делает стратегию активной.

По умолчанию: `false`

■ **spec.strategies.removePodsViolatingNodeAffinity.nodeAffinityType** массив строк

Определяет список используемых правил node affinity.

По умолчанию: `["requiredDuringSchedulingIgnoredDuringExecution"]`

■ **Элемент массива** строка

Допустимые значения: `requiredDuringSchedulingIgnoredDuringExecution`, `preferredDuringSchedulingIgnoredDuringExecution`

■ **spec.strategies.removePodsViolatingTopologySpreadConstraint** объект

Стратегия гарантирует, что поды, нарушающие ограничения распределения по топологии, будут вытеснены с узлов. Вытесняется минимальное количество подов, необходимое для приведения доменов топологии в соответствие с `maxSkew` каждого ограничения.

Полезна для перебалансировки подов между зонами доступности после восстановления зоны из аварии.

■ **spec.strategies.removePodsViolatingTopologySpreadConstraint.constraints** массив строк

Список типов ограничений распределения топологии, которые учитываются при принятии решения об вытеснении подов.

Ограничения распределения топологии управляют тем, как поды распределяются по топологии кластера (например, по регионам, зонам доступности, узлам).

По умолчанию учитываются только ограничения `DoNotSchedule` (жёсткие ограничения) — планировщик строго их соблюдает: под НЕ будет размещён, если это нарушит ограничение. `Descheduler` будет вытеснять поды, которые в данный момент нарушают такие ограничения (например, из-за отказа узлов или изменения топологии кластера после первоначального планирования).

Дополнительно можно включить тип `ScheduleAnyway` (мягкие ограничения) — планировщик старается их соблюдать, но не делает это строго: под БУДЕТ размещён, даже если ограничение не может быть выполнено, однако планировщик постарается минимизировать перекоп в распределении. Включение этого типа заставляет `descheduler` также вытеснять поды, нарушающие мягкие ограничения.

По умолчанию: `["DoNotSchedule"]`

■ **Элемент массива** строка

Допустимые значения: `DoNotSchedule`, `ScheduleAnyway`

■ **spec.strategies.removePodsViolatingTopologySpreadConstraint.enabled** булевый

Делает стратегию активной.

По умолчанию: `false`

■ **spec.strategies.removePodsViolatingTopologySpreadConstraint.topologyBalanceNodeFit** булевый

Если установлено значение `true`, стратегия будет учитывать, смогут ли поды быть размещены на других узлах при балансировке доменов топологии.

По умолчанию: `true`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описание одного экземпляра descheduler.

- **apiVersion**
строка
- **kind**
строка
- **metadata**
объект
 - **metadata.name**
строка
- **spec**
объект

Обязательный параметр

Описывает поведение экземпляра descheduler.

- **spec.deploymentTemplate**
объект

Описание шаблона Deployment'a descheduler.

- **spec.deploymentTemplate.nodeSelector**
объект
- **spec.deploymentTemplate.tolerations**
массив объектов
 - **spec.deploymentTemplate.tolerations.effect**
строка
 - **spec.deploymentTemplate.tolerations.key**
строка
 - **spec.deploymentTemplate.tolerations.operator**
строка
 - **spec.deploymentTemplate.tolerations.tolerationSeconds**
целочисленный
 - **spec.deploymentTemplate.tolerations.value**
строка

- **spec.deschedulerPolicy**
объект

`globalParameters` и `strategies` соответствуют документации descheduler.

- **spec.deschedulerPolicy.globalParameters**
объект

Параметры, применяемые ко всем политикам.

- **spec.deschedulerPolicy.globalParameters.evictFailedBarePods**
булевый

Позволяет вытеснять поды без `ownerReferences` и в состоянии `failed`.
- **spec.deschedulerPolicy.globalParameters.evictLocalStoragePods**
булевый

Позволяет вытеснять поды, использующие локальное хранилище.
- **spec.deschedulerPolicy.globalParameters.evictSystemCriticalPods**
булевый

Позволяет вытеснять поды любого приоритета (включая системные поды Kubernetes).
- **spec.deschedulerPolicy.globalParameters.ignorePvcPods**
булевый

Предотвращает вытеснение подов с PVC (Persistent Volume Claims).
- **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNamespace**
целочисленный

Ограничивает максимальное количество подов, которые могут быть вытеснены в пределах одного пространства имен.
- **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNode**
целочисленный

Ограничивает максимальное количество подов, которые могут быть вытеснены с одного узла.
- **spec.deschedulerPolicy.globalParameters.nodeSelector**
строка

- **spec.deschedulerPolicy.strategies**
объект

Список стратегий с соответствующими параметрами для данного экземпляра Descheduler.

- **spec.deschedulerPolicy.strategies.highNodeUtilization**
объект

Эта стратегия находит узлы с недостаточной загрузкой и вытесняет с них поды, предполагая, что эти поды будут компактно размещены на меньшем количестве узлов.
 - **spec.deschedulerPolicy.strategies.highNodeUtilization.enabled**
булевый

Обязательный параметр
 - **spec.deschedulerPolicy.strategies.highNodeUtilization.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.
- **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.
- **spec.deschedulerPolicy.strategies.highNodeUtilization.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.
- **spec.deschedulerPolicy.strategies.lowNodeUtilization**
объект

Эта стратегия находит узлы с низкой загрузкой и, если возможно, вытесняет с других узлов поды, предполагая, что новые экземпляры вытесненных подов будут размещены на этих менее загруженных узлах.
 - **spec.deschedulerPolicy.strategies.lowNodeUtilization.enabled**
булевый

Обязательный параметр
 - **spec.deschedulerPolicy.strategies.lowNodeUtilization.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.
 - **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.
 - **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.
 - **spec.deschedulerPolicy.strategies.lowNodeUtilization.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.
- **spec.deschedulerPolicy.strategies.removeDuplicates**
объект

Эта стратегия гарантирует, что на одном узле будет работать только один под, связанный с ReplicaSet (RS), ReplicationController (RC), StatefulSet или Job.
 - **spec.deschedulerPolicy.strategies.removeDuplicates.enabled**
булевый

Обязательный параметр
 - **spec.deschedulerPolicy.strategies.removeDuplicates.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.
 - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.
 - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.
 - **spec.deschedulerPolicy.strategies.removeDuplicates.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.
- **spec.deschedulerPolicy.strategies.removeFailedPods**
объект

Эта стратегия вытесняет поды, находящиеся в статусе `failed`.
 - **spec.deschedulerPolicy.strategies.removeFailedPods.enabled**
булевый

Обязательный параметр
 - **spec.deschedulerPolicy.strategies.removeFailedPods.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.
 - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.
 - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **spec.deschedulerPolicy.strategies.removeFailedPods.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts**
объект

Эта стратегия гарантирует, что поды с слишком большим количеством перезапусков будут удалены с узлов.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.enabled**
булевый

Обязательный параметр

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity**
объект

Эта стратегия гарантирует, что поды, которые не могут располагаться рядом с другими подами по определённому условию, будут удалены с узлов.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.enabled**
булевый

Обязательный параметр

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity**
объект

Эта стратегия гарантирует, что все поды которые не могут быть размещены на определённых узлах, в конечном итоге будут удалены с узлов.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.enabled**
булевый

Обязательный параметр

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.namespaceFilter**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter**
объект

Фильтрует узлы, к которым применяется стратегия.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFit**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.priorityFilter**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints**

объект

Эта стратегия гарантирует, что поды, нарушающие taint-лейблы с правилом `NoSchedule` на узлах, будут удалены.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.enabled`**
булевый

Обязательный параметр

- **`spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.namespaceFilter`**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter`**
объект

Фильтрует узлы, к которым применяется стратегия.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFit`**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.priorityFilter`**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint`**
объект

Эта стратегия гарантирует, что поды, нарушающие ограничения распределения по топологии (topology spread constraints), будут вытеснены с узлов.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.enabled`**
булевый

Обязательный параметр

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.namespaceFilter`**
объект

Ограничивает пространства имен, к которым применяется эта стратегия.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter`**
объект

Фильтрует узлы, к которым применяется стратегия.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFit`**
булевый

Если установлено значение `true`, descheduler будет учитывать, смогут ли поды, соответствующие критериям вытеснения, быть размещены на других узлах перед тем, как их вытеснить.

- **`spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.priorityFilter`**
объект

Только поды с приоритетом ниже этого значения будут вытеснены.

- **`status`**
объект

Most recently observed status of a descheduler instance.

- **`status.ready`**
булевый

Обязательный параметр

Примеры

Настройка интервала перераспределения подов

Чтобы настроить, как часто модуль `descheduler` будет запускать цикл перераспределения подов, используйте параметр `deschedulingInterval`.

Например, чтобы запускать `descheduler` каждые 5 минут, укажите `deschedulingInterval: Frequent`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true
  settings:
    deschedulingInterval: Frequent
```

Поддерживаются следующие значения:

- `Frequent` — запуск каждые 5 минут. Подходит для кластеров, где важно быстрее перераспределять поды;
- `Moderate` — запуск каждые 15 минут (по умолчанию);
- `Rare` — запуск каждые 30 минут. Подходит для кластеров, где важно минимизировать количество перераспределений подов.

Пример стратегии `LowNodeUtilization`

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

Пример стратегии HighNodeUtilization

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

Модуль documentation

i Веб-интерфейсы, связанные с модулем: documentation

Модуль `documentation` создает веб-интерфейс с документацией, соответствующей запущенной версии Deckhouse Kubernetes Platform.

Это может быть полезно, когда Deckhouse работает в сети с ограничением доступа в интернет.

Для получения адреса веб-интерфейса в шаблоне `publicDomainTemplate` глобального параметра конфигурации Deckhouse ключ `%s` замените на `documentation`.

Например, если `publicDomainTemplate` установлен как `%s-kube.company.my`, веб-интерфейс документации будет доступен по адресу `documentation-kube.company.my`.

Настройки

У модуля нет обязательных настроек.

В модуле реализовано 2 алерта.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `documentation`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot displays the Deckhouse web interface. On the left, a sidebar contains navigation links: 'Обзор', 'Обновления', 'Модули' (selected), 'Глобальные настройки', and 'Kubernetes'. The main panel is titled 'Система / Управление системой / Deckhouse' and 'Модули'. It features a search bar and a table of modules. The table has columns for 'Имя' (Name), 'Состояние' (Status), and 'Этап' (Stage). The 'admission-policy-engine' module is listed with a status of 'Включен' (Enabled). To the right, a detailed view of the 'admission-policy-engine' module is shown, with a toggle switch labeled 'Модуль включен' (Module enabled) which is currently turned on. Below this, there is a link to 'Документация к настройкам модуля admission-policy-engine'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `documentation`:

```
d8 system module enable documentation
```


- **С помощью ModuleConfig documentation.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `documentation` (создайте его, при необходимости).

Пример манифеста для включения модуля `documentation`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  enabled: true
```

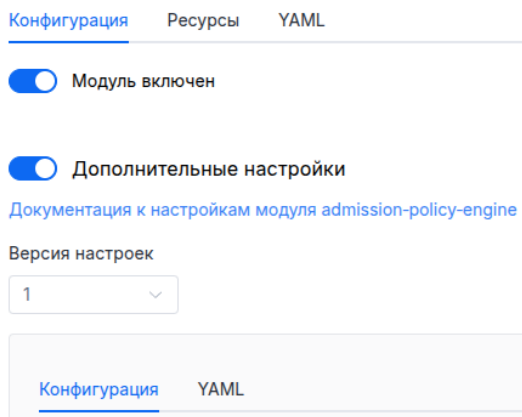
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `documentation`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `documentation` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `documentation`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `documentation` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc documentation
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `documentation`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `documentation`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.auth**
объект

Опции, связанные с аутентификацией и авторизацией доступа к веб-интерфейсу документации.

- **settings.auth.allowedUserEmails**
массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

■ **settings.auth.allowedUserGroups**
массив строк

Массив групп, пользователям которых разрешен доступ к просмотру документации.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

i **Внимание.** При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках `DexProvider`.

По умолчанию: `[]`

Примеры:

```
allowedUserGroups:
- admin
- users
```

```
allowedUserGroups: []
```

■ **settings.auth.externalAuthentication**
объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX `external-auth`, работающий на основе модуля Nginx `auth_request`).

i Внешняя аутентификация включается автоматически, если включен модуль `user-authn`.

■ **settings.auth.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

По умолчанию:

Пример:

```
authSignInURL: https://$host/dex-authenticator/sign_in
```

■ **settings.auth.externalAuthentication.authURL**
строка

URL сервиса аутентификации.

Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

По умолчанию:

Пример:

```
authURL: https://documentation-dex-authenticator.d8-system.svc.cluster.local/dex-authenticator/auth
```

○ **settings.https**
объект

Тип используемого сертификата.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
mode: Disabled
```

```
https:
mode: OnlyInURI
```

```
https:
mode: CustomCertificate
customCertificate:
secretName: foobar
```

```
https:
mode: CertManager
certManager:
clusterIssuerName: letsencrypt
```

■ **settings.https.certManager**
объект

Настройки для `certmanager`.

- **settings.https.certManager.clusterIssuerName**

строка

Тип ресурса ClusterIssuer, используемого для заказа SSL-сертификата (доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но можно определить свои).

По умолчанию: `letsencrypt`

Пример:

```
clusterIssuerName: letsencrypt
```

- **settings.https.customCertificate**

объект

Настройки для использования пользовательского сертификата.

- **settings.https.customCertificate.secretName**

строка

Имя ресурса Secret в пространстве имен `d8-system`, который будет использоваться для веб-интерфейса документации.

Secret должен быть в формате `kubernetes.io/tls`.

- **settings.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — доступ по HTTPS с заказом сертификата согласно ClusterIssuer'y, заданному в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — доступ по HTTPS с использованием сертификата из пространства имен `d8-system`.
- `Disabled` — доступ только по HTTP.
- `OnlyInURI` — доступ по HTTP с внешним HTTPS-балансировщиком. Балансировщик терминирует HTTPS, и все ссылки в `user-authn` генерируются с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `CertManager`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**

строка

Класс Ingress-контроллера веб-интерфейса документации.

Оptionальный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, `nodeSelector` будет определяться автоматически.

Пример:

```
nodeSelector:
  disktype: ssd
```

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, `tolerations` будет определяться автоматически.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**

строка

- **settings.tolerations.operator**

строка

- **settings.tolerations.tolerationSeconds**

целочисленный

- **settings.tolerations.value**

строка

Аутентификация

По умолчанию используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication` (см. ниже). Если эти варианты отключены, модуль включит базовую аутентификацию со сгенерированным паролем.

Чтобы посмотреть сгенерированный пароль, выполните команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values documentation -O json | jq
'.internal.auth.password'
```

Чтобы сгенерировать новый пароль, удалите ресурс Secret:

```
d8 k -n d8-system delete secret/documentation-basic-auth
```

i Параметр `auth.password` больше не поддерживается.

Примеры

Пример конфигурации модуля

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system
  externalAuthentication:
    authURL: "https://<applicationDomain>/auth"
    authSignInURL: "https://<applicationDomain>/sign-in"
    authResponseHeaders: "Authorization"
```

Модуль extended-monitoring

Модуль `extended-monitoring` расширяет возможности мониторинга кластера за счёт дополнительных Prometheus exporter'ов, которые позволяют выявлять потенциальные проблемы до того, как они скажутся на работе сервисов.

Возможности модуля:

- Расширенный сбор метрик — собирает дополнительные метрики, а также включает готовые алерты и дашборды, которые позволяют быстрее обнаруживать и диагностировать инциденты:
 - собирает и экспортирует метрики по свободному месту и inode на узлах, а также по объектам с лейблом `extended-monitoring.deckhouse.io/enabled=""` в пространстве имён;
 - автоматически формирует алерты при достижении пороговых значений.
- Мониторинг контейнерных образов:
 - добавляет метрики и отправляет алерты о недоступности образов контейнеров в registry для всех типов рабочей нагрузки (Deployments, StatefulSets, DaemonSets, CronJobs);
 - помогает заранее узнать о возможных проблемах с запуском или обновлением подов.
- События в кластере — собирает события Kubernetes и отображает их в виде метрик, что позволяет отслеживать динамику изменений и быстрее реагировать на инциденты.
- Контроль сертификатов:
 - сканирует Secret'ы кластера и генерирует метрики об истечении срока действия x509-сертификатов;
 - позволяет не пропускать критические моменты и вовремя обновлять сертификаты, избегая простоя приложений из-за просроченных сертификатов.

Настройки

Как использовать extended-monitoring-exporter

Чтобы включить экспортирование `extended-monitoring` метрик, нужно указать в пространстве имён лейбл `extended-monitoring.deckhouse.io/enabled` любым удобным способом, например:

- добавить в проект соответствующий helm-чарт (рекомендуемый);
- добавить в описание `.gitlab-ci.yml` (`kubectl patch/create`);
- добавить вручную (`d8 k label namespace my-app-production extended-monitoring.deckhouse.io/enabled=""`);
- настроить через `namespace-configurator` модуль.

Сразу же после этого для всех поддерживаемых Kubernetes-объектов в данном пространстве имён в Prometheus появятся `default`-метрики + любые кастомные с префиксом `threshold.extended-monitoring.deckhouse.io/`. Для ряда `non-namespaced` Kubernetes-объектов, описанных ниже, мониторинг включается автоматически.

К Kubernetes-объектам `threshold.extended-monitoring.deckhouse.io/` что-то свое можно добавить любые другие лейблы с указанным значением. Пример: `d8 k label pod test threshold.extended-monitoring.deckhouse.io/disk-inodes-warning=30`. В таком случае значение из лейбла заменит значение по умолчанию.

Если вы хотите переопределить значения `threshold` для всех объектов в определенном пространстве имен, вы можете установить лейбл `threshold.extended-monitoring.deckhouse.io/` на уровне namespace. Например: `d8 k label namespace my-app-production threshold.extended-monitoring.deckhouse.io/5xx-warning=20`. Это заменит значение по умолчанию для всех объектов namespace, для которых еще не установлен этот лейбл.

Слежение за объектом можно отключить индивидуально, поставив на него лейбл `extended-monitoring.deckhouse.io/enabled=false`. Соответственно, отключатся и лейблы по умолчанию, а также все алерты, привязанные к лейблам.

Стандартные лейблы и поддерживаемые Kubernetes-объекты

Далее приведен список используемых в Prometheus Rules лейблов, а также их стандартные значения.

Обратите внимание, что все лейблы начинаются с префикса `threshold.extended-monitoring.deckhouse.io/`. Указанное в лейбле значение — число, которое устанавливает порог срабатывания алерта.

Например, лейбл `threshold.extended-monitoring.deckhouse.io/5xx-warning: "5"` на Ingress-ресурсе изменяет порог срабатывания алерта с 10% (по умолчанию) на 5%.

Non-namespaced Kubernetes-объекты

Non-namespaced Kubernetes-объекты, то есть объекты вне пространств имён, не нуждаются в лейблах на этих пространствах и мониторинг на них включается по умолчанию при включении модуля.

Узел

Лейбл	Тип	Значение по умолчанию
disk-bytes-warning	int (percent)	70
disk-bytes-critical	int (percent)	80
disk-inodes-warning	int (percent)	90
disk-inodes-critical	int (percent)	95
load-average-per-core-warning	int	3
load-average-per-core-critical	int	10

Важно! Эти лейблы **не** действуют для тех разделов, в которых расположены `imagefs` (по умолчанию — `/var/lib/docker`) и `nodefs` (по умолчанию — `/var/lib/kubelet`). Для этих разделов пороги настраиваются полностью автоматически согласно `eviction thresholds` в `kubelet`. Значения по умолчанию указаны в документации Kubernetes, в описании экспортера.

Namespaced Kubernetes-объекты

Под

Лейбл	Тип	Значение по умолчанию
disk-bytes-warning	int (percent)	85
disk-bytes-critical	int (percent)	95
disk-inodes-warning	int (percent)	85
disk-inodes-critical	int (percent)	90

Ingress

Лейбл	Тип	Значение по умолчанию
5xx-warning	int (percent)	10
5xx-critical	int (percent)	20

Deployment

Лейбл	Тип	Значение по умолчанию
replicas-not-ready	int (count)	0

Порог подразумевает количество недоступных реплик **сверх** `maxUnavailable`. Сработает, если недоступно реплик больше на указанное значение, чем разрешено в `maxUnavailable`. То есть при нуле сработает, если недоступно больше, чем указано в `maxUnavailable`, а при единице сработает, если недоступно больше, чем указано в `maxUnavailable`, плюс 1. Таким образом, у конкретных Deployment, которые находятся в пространстве имён со включенным расширенным мониторингом и которым допустимо быть недоступными, можно установить этот параметр, чтобы не получать ненужные алерты.

StatefulSet

Лейбл	Тип	Значение по умолчанию
replicas-not-ready	int (count)	0

Порог подразумевает количество недоступных реплик **сверх** `maxUnavailable` (см. комментарии к Deployment).

DaemonSet

Лейбл	Тип	Значение по умолчанию
replicas-not-ready	int (count)	0

Порог подразумевает количество недоступных реплик **сверх** maxUnavailable (см. комментарии к Deployment).

CronJob

Работает только выключение через лейбл `extended-monitoring.deckhouse.io/enabled=false`.

Как работает

Модуль экспортирует в Prometheus специальные лейблы Kubernetes-объектов. Позволяет улучшить Prometheus-правила путем добавления порога срабатывания для алертов. Использование метрик, экспортируемых данным модулем, позволяет, например, заменить «магические» константы в правилах.

До:

```
(
  kube_statefulset_status_replicas - kube_statefulset_status_replicas_ready
)
> 1
```

После:

```
(
  kube_statefulset_status_replicas - kube_statefulset_status_replicas_ready
)
> on (namespace, statefulset)
(
  max by (namespace, statefulset) (extended_monitoring_statefulset_threshold{threshold="replicas-not-ready"})
)
```

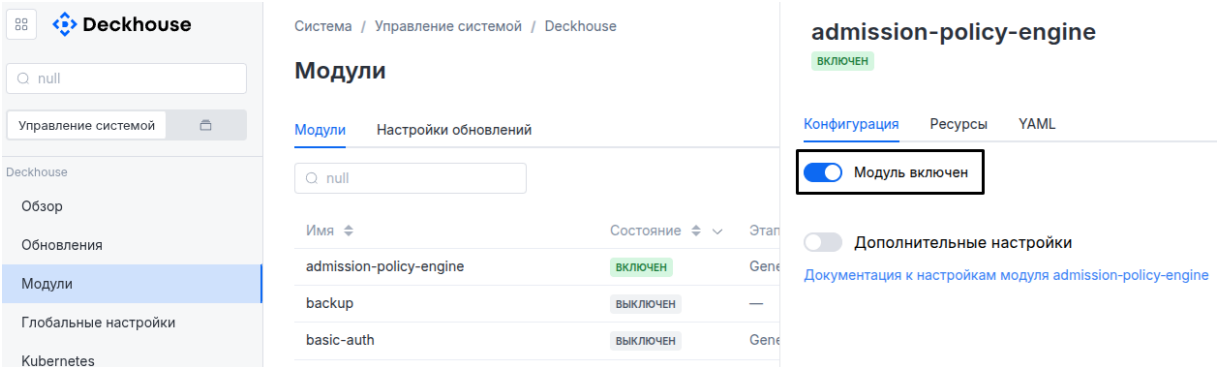
В модуле реализовано 58 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `extended-monitoring`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `extended-monitoring`:

```
d8 system module enable extended-monitoring
```

- **С помощью ModuleConfig `extended-monitoring`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `extended-monitoring` (создайте его, при необходимости).

Пример манифеста для включения модуля `extended-monitoring`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: extended-monitoring
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `extended-monitoring`,

включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `extended-monitoring` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `extended-monitoring`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `extended-monitoring` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc extended-monitoring
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `extended-monitoring`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `extended-monitoring`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: extended-monitoring
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Если для поля `.imageAvailability.skipRegistryCertVerification` установлено значение `true`, добавьте поле `.imageAvailability.registry.tlsConfig.insecureSkipVerify=true`. После этого удалите поле `.imageAvailability.skipRegistryCertVerification` из объекта. Если в результате объект `.imageAvailability` становится пуст, удалите его.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.certificates**
объект

Настройки для мониторинга сертификатов в кластере Kubernetes.

- **settings.certificates.exporterEnabled**
булевый

Включен ли x509-certificate-exporter.

По умолчанию: `false`

- **settings.events**
объект

Настройки для мониторинга event'ов в кластере Kubernetes.

- **settings.events.exporterEnabled**
булевый

Включен ли eventsExporter.

По умолчанию: `false`

- **settings.events.severityLevel**
строка

Enables eventsExporter.

По умолчанию: `OnlyWarnings`

Допустимые значения: `All`, `OnlyWarnings`

- **settings.imageAvailability**
объект

Настройки для мониторинга доступности образов в кластере.

- **settings.imageAvailability.defaultRegistry**
строка

Стандартный registry для проверки доступности образов. Используется если в образе не указан registry.

Если не задан, то используется `index.docker.io`

По умолчанию: `index.docker.io`

- **settings.imageAvailability.exporterEnabled**
булевый

Включен ли imageAvailabilityExporter.

По умолчанию: `true`

- **settings.imageAvailability.forceCheckDisabledControllers**
массив строк

Список контроллеров, которые необходимо проверять, даже если количество реплик подов равняется 0 или контроллер находится в статусе `suspend` (приостановленный).

Укажите `All`, для проверки всех типов контроллеров.

Пример:

```
forceCheckDisabledControllers:
- Deployment
- StatefulSet
```

- **Элемент массива**
строка

Допустимые значения: `Deployment`, `StatefulSet`, `DaemonSet`, `CronJob`, `All`

- **settings.imageAvailability.ignoredImages**
массив строк

Список образов, для которых нужно пропустить проверку, например `alpine:3.12` или `quay.io/test/test:v1.1`.

Пример:

```
ignoredImages:
- alpine:3.10
- alpine:3.2
```

- **settings.imageAvailability.imageCheckInterval**
строка

Интервал запросов в registry для проверки доступности образа.

Поддерживаемый формат - Go-style duration ("`10s`", "`10m`", "`10h`").

Если не задан, то используется значение по умолчанию в 60 секунд.

По умолчанию: `60s`

- **settings.imageAvailability.mirrors**
массив объектов

Список зеркал для container registry.

Пример:

```
mirrors:
- original: docker.io
  mirror: mirror.gcr.io
- original: internal-registry.com
  mirror: mirror.internal-registry.com
```

- **settings.imageAvailability.mirrors.mirror**
строка

Обязательный параметр

- **settings.imageAvailability.mirrors.original**
строка

Обязательный параметр

- **settings.imageAvailability.registry**
объект

Настройки подключения к container registry.

- **settings.imageAvailability.registry.scheme**
строка

Протокол доступа к container registry.

По умолчанию: `HTTPS`

Допустимые значения: `HTTP`, `HTTPS`

- **settings.imageAvailability.registry.tlsConfig**
объект

Настройки подключения к container registry

- **settings.imageAvailability.registry.tlsConfig.ca**
строка

Корневой сертификат, которым можно проверить сертификат container registry при работе по HTTPS (если registry использует самоподписанные SSL-сертификаты).

- **settings.imageAvailability.registry.tlsConfig.insecureSkipVerify**
булевый

Пропускать ли валидацию TLS-сертификата для container registry.

По умолчанию: `false`

- **settings.nodeSelector**
объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations**
массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Extended monitoring модуль: FAQ

Как переключиться на HTTP вместо HTTPS для проверки образов из собственного registry?

Чтобы изменить протокол проверки вашего registry с HTTPS на HTTP, измените параметр `settings.imageAvailability.registry.scheme` в конфигурации модуля.

Подробные инструкции смотрите в документации по настройке модуля.

Модуль ingress-nginx

Устанавливает и управляет Ingress NGINX Controller с помощью кастомных ресурсов. Если узлов для размещения Ingress-контроллера больше одного, он устанавливается в отказоустойчивом режиме и учитывает все особенности реализации инфраструктуры облаков и bare metal, а также кластеров Kubernetes различных типов.

Поддерживает запуск и раздельное конфигурирование одновременно нескольких Ingress-контроллеров — один **основной** и неограниченное количество **дополнительных**. Например, это позволяет отделять внешние и intranet Ingress-ресурсы приложений.

i

В 2025 году Ingress NGINX был переведён в режим сопровождения без планов активного развития новых возможностей. Дальнейшее развитие средств балансировки входящего трафика в Kubernetes ориентировано на Gateway API.

На поддержку модуля в составе Deckhouse Kubernetes Platform это не распространяется: модуль сопровождается командой Deckhouse, включая обновления безопасности. Подробнее см. раздел «Поддержка и безопасность модуля».

Текущая используемая версия nginx - 1.30.3.

Маршрутизация трафика

Трафик к `ingress-nginx` может быть отправлен несколькими способами:

- напрямую без внешнего балансировщика;
- через внешний LoadBalancer, в том числе поддерживаются:
 - Qrator,

- Cloudflare,
- AWS LB,
- GCE LB,
- ACS LB,
- Yandex LB,
- OpenStack LB.

Поддерживаемые протоколы

Модуль позволяет использовать следующие протоколы для приема и дальнейшей маршрутизации трафика:

- HTTP/HTTPS: основной протокол маршрутизация трафика, допускается выбор используемых версий между HTTP/1.1, HTTP/2 и HTTP/3;
- WebSocket: поддерживается nginx нативно, без дополнительной настройки со стороны пользователя;
- gRPC: поддерживается, для использования необходимо указать соответствующую аннотацию;
- CGI: поддерживается, для использования необходимо указать соответствующую аннотацию;
- Proxy Protocol: поддерживается организация приема Proxy Protocol-трафика засчет использования соответствующих инлетов;
- SSL Passthrough: поддерживается сквозная маршрутизация SSL-трафика засчет использования соответствующих инлетов.

На данный момент в нашей реализации ingress-nginx не поддерживается маршрутизация TCP/UDP трафика (функционал tcp-services/udp-services). Для маршрутизации TCP/UDP трафика предлагается применять нативное решение Kubernetes NodePort Service или рассмотреть использование модуля alb для организации маршрутизации трафика на основе Gateway API (доступен к использованию начиная с версии DKP 1.76).

Использование HTTPS

Модуль позволяет управлять политиками безопасности HTTPS каждого Ingress NGINX Controller, в том числе:

- параметрами HSTS;
- набором доступных версий SSL/TLS и протоколов шифрования.

По умолчанию во всех версиях контроллера используется TLSv1.2 и TLSv1.3 с возможностью ручной настройки устаревших версий. В случае использования TLSv1.2 применяются следующие протоколы шифрования:

- обмен ключами: ECDHE, DHE;
- аутентификация: ECDSA, RSA;
- шифрование (AEAD): AES-GCM (128/256), ChaCha20-Poly1305;
- целостность данных (MAC): SHA256, SHA384.

При использовании TLSv1.3 набор протоколов шифрования определяется библиотекой OpenSSL (версии 3.3.3+).

Также модуль ingress-nginx интегрирован с модулем cert-manager, при взаимодействии с которым возможны автоматический заказ SSL-сертификатов и дальнейшее использование сертификатов Ingress NGINX Controller.

Мониторинг и статистика

В нашей реализации ingress-nginx добавлена система сбора статистики в Prometheus с множеством метрик:

- по длительности времени всего ответа и апстрима отдельно;
- кодам ответа;
- количеству повторов запросов (retry);
- размерам запроса и ответа;
- методам запросов;
- типам content-type;
- географии распределения запросов и т. д.

Данные доступны в нескольких параметрах:

- по namespace;
- vhost;
- ingress-ресурсу;
- location (в nginx).

Все графики собраны в виде удобных дашбордов в Grafana, при этом есть возможность drill-down'a по графикам. Например, при просмотре статистики пространств имён, можно нажать ссылку на дашборд в Grafana, углубиться в статистику по vhosts в соответствующем namespace и т.д.

Статистика

Основные принципы сбора статистики

1. На каждый запрос на стадии log_by_lua_block вызывается модуль, который рассчитывает необходимые данные и складывает их в буфер (у каждого nginx worker собственный буфер).
2. На стадии init_by_lua_block для каждого nginx worker запускается процесс, который раз в секунду асинхронно отправляет данные в формате protobuf через TCP socket в protobuf_exporter (разработка DKP).
3. protobuf_exporter запущен sidecar-контейнером в поде с ingress-controller, принимает сообщения в формате protobuf, разбирает, агрегирует их по установленным правилам и экспортирует в формате для Prometheus.
4. Prometheus каждые 30 секунд собирает метрики как в ingress-controller (там есть небольшое количество нужных нам метрик), так и в protobuf_exporter, на основании этих данных все работает!

Какая статистика собирается и как она представлена

У всех собираемых метрик есть служебные лейблы, позволяющие идентифицировать экземпляр контроллера: controller, app, instance и endpoint (они видны в /prometheus/targets).

- Все метрики (кроме geo), экспортируемые protobuf_exporter, представлены в трех уровнях детализации:
 - ingress nginx overall * — общий обзор, у всех метрик есть лейблы namespace, vhost и content kind;
 - ingress nginx detail * — кроме лейблов уровня overall, добавляются ingress, service, service_port и location;
 - ingress nginx detail backend * — ограниченная часть данных, собирается в разрезе по бэкендам. У этих метрик, кроме лейблов уровня detail, добавляется лейбл pod_ip.
- Для уровней overall и detail собираются следующие метрики:
 - * requests total — counter количества запросов (дополнительные лейблы — scheme, method);
 - * responses total — counter количества ответов (дополнительный лейбл — status);
 - * _request_seconds_{sum,count,bucket} — histogram времени ответа;

- `* bytes_received {sum,count,bucket}` — histogram размера запроса;
- `* bytes_sent {sum,count,bucket}` — histogram размера ответа;
- `* upstream_response_seconds {sum,count,bucket}` — histogram времени ответа upstream'a (используется сумма времен ответов всех upstream'ов, если их было несколько);
- `* lowres_upstream_response_seconds {sum,count,bucket}` — то же самое, что и предыдущая метрика, только с меньшей детализацией (подходит для визуализации, но не подходит для расчета quantile);
- `* upstream_retries {count,sum}` — количество запросов, при обработке которых были retry бэкендов, и сумма retry'ев.
- Для уровня `overall` собираются следующие метрики:
 - `* geohash_total` — counter количества запросов с определенным geohash (дополнительные лейблы — `geohash`, `place`).
- Для уровня `detail_backend` собираются следующие метрики:
 - `* lowres_upstream_response_seconds` — то же самое, что аналогичная метрика для `overall` и `detail`;
 - `* responses_total` — counter количества ответов (дополнительный лейбл — `status_class`, а не просто `status`);
 - `* upstream_bytes_received_sum` — counter суммы размеров ответов бэкенда.

Поддержка и безопасность модуля

Модуль `ingress-nginx` входит в сопровождение Deckhouse Kubernetes Platform на весь срок поддержки платформы, вне зависимости от режима развития upstream-проекта. Команда Deckhouse отслеживает CVE в контроллере и зависимостях — NGINX, Lua-модули, базовые образы — и предоставляет исправления в релизах платформы. Для соответствия ожиданиям PCI DSS по вендорской поддержке и срокам устранения уязвимостей ответственным вендором модуля является компания «Флант». Сертификация DKP в ФСТЭК России фиксирует, в том числе, процессы управления уязвимостями и выпуск обновлений безопасности.

Настройки

i Если модуль был выключен и необходимо его включить, обратите внимание на глобальный параметр `publicDomainTemplate`. Укажите его, если он не указан, иначе Ingress-ресурсы для служебных компонентов DKP (`dashboard`, `user-auth`, `grafana`, `upmeter` и т. п.) не будут созданы.

Конфигурация Ingress-контроллеров выполняется с помощью Custom Resource `IngressNginxController`.

В модуле реализовано 14 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `ingress-nginx`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `ingress-nginx`:

```
d8 system module enable ingress-nginx
```

- **С помощью ModuleConfig ingress-nginx.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig ingress-nginx` (создайте его, при необходимости).

Пример манифеста для включения модуля `ingress-nginx`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: ingress-nginx
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `ingress-nginx`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или

укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация

Ресурсы

YAML

Модуль включен

Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация

YAML

Вы также можете отредактировать объект ModuleConfig `ingress-nginx` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `ingress-nginx`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `ingress-nginx` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc ingress-nginx
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `ingress-nginx`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `ingress-nginx`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: ingress-nginx
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.defaultControllerVersion**
строка

Версия контроллера `ingress-nginx`, которая будет использоваться для всех контроллеров по умолчанию, если не был задан параметр `controllerVersion` в `IngressNginxController` CR. Поддерживаемые версии: `1.10`, `1.12`, `1.14`.

По умолчанию: `1.10`

Допустимые значения: `1.10`, `1.12`, `1.14`

- **settings.highAvailability**
булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

- **settings.storageClass**
строка

Имя StorageClass, который будет использоваться для хранения баз GeoIP (StatefulSet `geoproxy`).

Если значение не указано, используется StorageClass существующей PVC. Если PVC ещё нет, будет использован StorageClass согласно настройке глобального параметра `storageClass`.

Глобальный параметр `storageClass` учитывается только при включении модуля. Изменение параметра при включённом модуле не приводит к автоматическому перезаказу диска.

При значении `false` будет использоваться `emptyDir`.

Примеры:

```
storageClass: ceph-ssd
```

```
storageClass: 'false'
```

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

IngressNginxController

Scope: Cluster

- v1
- v1alpha1
- **status**
объект
 - **status.loadBalancer**
объект
 - **status.loadBalancer.hostname**
строка
DNS-имя балансировщика нагрузки.
 - **status.loadBalancer.ip**
строка
IP-адрес балансировщика нагрузки.

- **spec**
объект

Обязательный параметр

- **spec.acceptRequestsFrom**
массив строк

Список адресов в формате CIDR, которым разрешен доступ к контроллеру.

Независимо от инлета всегда проверяется непосредственный адрес (поле `original address` в логах), с которого производится подключение, а не адрес клиента, который может передаваться в некоторых инлетах через заголовки или с использованием `proxy protocol`.

Параметр реализован с помощью `map module`. Если адрес, с которого производится подключение, не разрешен, `nginx` закрывает соединение, отправляя код `444`.

i По умолчанию к контроллеру можно подключаться с любых адресов.

- **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.additionalHeaders**
объект

Дополнительные заголовки, которые будут добавлены к каждому запросу. Указываются в формате `ключ: значение(строка)`.

- **spec.additionalLogFields**
объект

Дополнительные поля, которые будут добавлены в логи `nginx`. Указываются в формате `ключ: значение(строка)`.

- **spec.annotationValidationEnabled**
булевый

Включает валидацию аннотаций Ingress-правил.

По умолчанию: `false`

- **spec.chaosMonkey**
булевый

Инструмент, позволяющий систематически вызывать случайные прерывания работы подов контроллера.

Предназначен для проверки Ingress-контроллера на реальную работу отказоустойчивости.

По умолчанию: `false`

- **spec.config**
объект

Секция настроек Ingress-контроллера, в которую в формате `ключ: значение(строка)` можно записать любые возможные опции.

i Ошибка в опциях может привести к неработоспособности Ingress-контроллера.

Используйте опцию с осторожностью, поскольку обратная совместимость и работоспособность Ingress-контроллера не гарантируются.

- **spec.controllerLogLevel**

строка

Определяет уровень логирования для журналов Ingress-контроллера.

По умолчанию: `Info`

Допустимые значения: `Error` — в журнал будут записываться только критические ошибки.

`Warn` — в журнал будут записываться предупреждения и ошибки.

`Info` — информационные сообщения, включая базовые сведения для отладки.

`Extended` — расширенная информация об изменениях в состоянии системы.

`Debug` — подробный отладочный вывод, может включать избыточные данные.

`Trace` — журналы на уровне трассировки с подробным пошаговым контекстом.

- **spec.controllerPodsAdditionalAnnotations**

объект

Дополнительные пользовательские аннотации для подов Ingress-контроллера.

i Используйте параметр с осторожностью, поскольку обратная совместимость и работоспособность Ingress-контроллера при использовании параметра не гарантируются.

- **spec.controllerVersion**

строка

Версия Ingress NGINX Controller.

По умолчанию используется версия из настроек модуля. Поддерживаемые версии: `1.10`, `1.12`, `1.14`.

Допустимые значения: `1.10`, `1.12`, `1.14`

- **spec.customErrors**

объект

Секция с настройкой кастомизации HTTP-ошибок.

i Если секция указана в настройках, все параметры в ней являются обязательными.

Изменение любого параметра приводит к перезапуску всех Ingress-контроллеров.

- **spec.customErrors.codes**

массив строк

Обязательный параметр

Список кодов ответа (массив), при которых запрос будет перенаправляться на кастомный бэкэнд.

- **Элемент массива**

строка

Шаблон: `^[1-5][0-9][0-9]$`

- **spec.customErrors.namespace**

строка

Обязательный параметр

Название пространства имён, в котором будет находиться сервис, используемый в качестве кастомного бэкэнда по умолчанию.

Пример:

```
namespace: default
```

- **spec.customErrors.serviceName**

строка

Обязательный параметр

Имя сервиса, который будет использоваться в качестве кастомного бэкэнда по умолчанию.

Пример:

```
serviceName: custom-errors-backend-service
```

- **spec.defaultSSLCertificate**

объект

Сертификат, который используется:

- при запросах на `catch-all`-сервер (подразумевается директива `server nginx`). На `catch-all`-сервер попадают запросы, для которых нет соответствующего Ingress-ресурса;
- для Ingress-ресурсов, в которых не задан `secretName` в секции `tls`.

По умолчанию используется самоподписанный сертификат.



Внимание. Параметр не влияет на сертификаты, используемые в Ingress-ресурсах модулей Deckhouse. Чтобы указать сертификат для Ingress-ресурсов модулей Deckhouse, используйте глобальный параметр `modules.https.customCertificate`.

- **`spec.defaultSSLCertificate.secretRef`**
объект

Ссылка на Secret для передачи Ingress-контроллеру.

- **`spec.defaultSSLCertificate.secretRef.name`**
строка

Имя Secret, содержащего SSL-сертификат.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **`spec.defaultSSLCertificate.secretRef.namespace`**
строка

Название пространства имён, в котором находится Secret с SSL-сертификатом.

По умолчанию: `d8-ingress-nginx`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **`spec.disableHTTP2`**

булевый

Выключает HTTP/2.

По умолчанию: `false`

- **`spec.enableHTTP3`**

булевый

Включает HTTP/3.



HTTP/3 нельзя включить, если выбран инлет `HostPortWithProxyProtocol` или `LoadBalancerWithProxyProtocol`.

В случае инлета `HostWithFailover` HTTP/3 включается только для основного контроллера; резервный контроллер всегда работает без HTTP/3.

По умолчанию: `false`

- **`spec.enableIstioSidecar`**

булевый

Добавляет к подам контроллера аннотации для автоматического инжекта сайдкаров Istio.

При включении этого параметра к подам Ingress-контроллера добавляются аннотации `sidecar.istio.io/inject: "true"` и `traffic.sidecar.istio.io/includeOutboundIPRanges: "<Service CIDR>"`. При создании таких подов к ним автоматически будут добавлены сайдкары Istio с помощью mutating webhook. После этого весь трафик в сторону Service CIDR будет перехватываться сайдкаром.

Чтобы воспользоваться этой функцией, добавьте аннотации в прикладные Ingress-ресурсы:

- `nginx.ingress.kubernetes.io/service-upstream: "true"` — с этой аннотацией Ingress-контроллер будет отправлять запросы на ClusterIP сервиса (из диапазона Service CIDR) вместо того, чтобы слать их напрямую в поды приложения. Сайдкар `istio-proxy` перехватывает трафик только в сторону диапазона Service CIDR, остальные запросы отправляются напрямую;
- `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — с данной аннотацией сайдкар сможет идентифицировать прикладной сервис, для которого предназначен запрос.



Внимание. Данный параметр не может быть включен, если инлет Ingress-контроллера установлен в значение `HostWithFailover`.

- **`spec.geolIP2`**

объект

Опции для включения GeolIP2.

- **`spec.geolIP2.maxmindAccountID`**
целочисленный

Идентификатор учетной записи MaxMind, используемый для аутентификации загрузки базы данных GeoIP2.

Требуется, когда указан `maxmindLicenseKey`. Если задан `maxmindAccountID`, загрузка выполняется официальной библиотекой `geoipupdate`, что позволяет пропускать неизменённые обновления и экономить лимит лицензии.

Допустимые значения: `1 <= X`

- **spec.geoIP2.maxmindEditionIDs**

массив строк

Список ревизий баз данных, которые будут скачаны при старте.

Подробнее о базах данных GeoLite — в статье в блоге MaxMind.

По умолчанию: `["GeoLite2-City", "GeoLite2-ASN"]`

- **Элемент массива**

строка

Допустимые значения: `GeoIP2-Anonymous-IP`, `GeoIP2-Country`, `GeoIP2-City`, `GeoIP2-Connection-Type`, `GeoIP2-Domain`, `GeoIP2-ISP`, `GeoIP2-ASN`, `GeoLite2-ASN`, `GeoLite2-Country`, `GeoLite2-City`

- **spec.geoIP2.maxmindLicenseKey**

строка

Лицензионный ключ для скачивания базы данных GeoIP2.

При наличии ключа в конфигурации база GeoIP2 скачивается при каждом старте контроллера. Подробнее о получении ключа — в статье в блоге MaxMind.

- **spec.geoIP2.maxmindMirror**

объект

Настройки зеркала для скачивания баз GeoIP. Если параметр не указан или имеет пустое значение, то по умолчанию будет использоваться `https://download.maxmind.com`.

Пример корректного URL: `https://mirror.local/GeoLite2-City.tar.gz`.

Если зеркало используется для хранения предзагруженных архивов, параметр `licenseKey` можно не указывать.

- **spec.geoIP2.maxmindMirror.ca**

строка

Пользовательский корневой сертификат в формате PEM для проверки TLS-соединения с зеркалом.

Пример: `—BEGIN CERTIFICATE— ... —END CERTIFICATE—`

- **spec.geoIP2.maxmindMirror.insecureSkipVerify**

булевый

Отключает проверку TLS сертификатов зеркала, например, если используется самоподписанный сертификат.

- **spec.geoIP2.maxmindMirror.url**

строка

Обязательный параметр

URL для скачивания баз GeoIP.

Шаблон: `^https?://.+`

Примеры:

```
url: https://mirror.local
```

```
url: absolute path https://mirror.local/GeoLite2-City.tar.gz
```

- **spec.hostPort**

объект

Секция настроек инлета `HostPort`.

- **spec.hostPort.acceptClientIPHeadersFrom**

массив строк

Определяет список доверенных подсетей в формате CIDR, запросы от которых содержат корректные заголовки `X-Forwarded-*/ProxyProtocol`.

По умолчанию: `["0.0.0.0/0"]`

Пример:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

- **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.hostPort.behindL7Proxy**
булевый

Включает обработку и передачу заголовков `X-Forwarded-*`.

i **Внимание.** Перед использованием этой опции убедитесь, что запросы к Ingress направляются только от доверенных источников. Для настройки ограничений воспользуйтесь параметром `acceptRequestsFrom`.

- **spec.hostPort.httpPort**
целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, подключение по HTTP невозможно.

i Параметр обязателен в случае, если не задан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPort.httpsPort**
целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, подключение по HTTPS невозможно.

i Параметр обязателен в случае, если не задан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.hostPort.realIPHeader**
строка

Заголовок, из которого будет получен настоящий IP-адрес клиента.

i Работает только при включенной опции `behindL7Proxy`.

По умолчанию: `X-Forwarded-For`

Пример:

```
realIPHeader: CF-Connecting-IP
```

- **spec.hostPortWithProxyProtocol**
объект

Секция настроек инлета `HostPortWithProxyProtocol`.

- **spec.hostPortWithProxyProtocol.acceptClientIPHeadersFrom**
массив строк

Определяет список доверенных подсетей в формате CIDR, запросы от которых содержат корректные заголовки `X-Forwarded-*/ProxyProtocol`.

По умолчанию: `["0.0.0.0/0"]`

Пример:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0-9))$`

- **spec.hostPortWithProxyProtocol.httpPort**

целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, подключение по HTTP невозможно.



Параметр обязателен в случае, если не задан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPortWithProxyProtocol.httpsPort**

целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, подключение по HTTPS невозможно.



Параметр обязателен в случае, если не задан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.hostPortWithSSLPassthrough**

объект

Секция настроек инлета `HostPortWithSSLPassthrough`.

- **spec.hostPortWithSSLPassthrough.httpPort**

целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, подключение по HTTP невозможно.



Параметр обязателен в случае, если не задан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPortWithSSLPassthrough.httpsPort**

целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, подключение по HTTPS невозможно.



Параметр обязателен в случае, если не задан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.hsts**

булевый

Включает использование заголовков ответа HTTP Strict-Transport-Security (HSTS). Подробнее о заголовках HSTS — в статье на портале MDN Web Docs.

По умолчанию: `false`

- **spec.hstsOptions**

объект

Параметры HSTS.

- **spec.hstsOptions.includeSubDomains**

булевый

Применяет настройки HSTS ко всем поддоменам сайта.

По умолчанию: `false`

- **spec.hstsOptions.maxAge**

строка

Время в секундах, в течение которого браузер помнит, что сайт доступен только с помощью HTTPS.

По умолчанию: `31536000`

Шаблон: `^[1-9][0-9]*$`

Пример:

```
maxAge: '31536000'
```

- **spec.hstsOptions.preload**

булевый

Добавляет сайт в список предзагрузки.

Список указывает браузерам, что подключение к перечисленным сайтам должно осуществляться только по HTTPS.

По умолчанию: `false`

- **spec.ingressClass**

строка

Имя Ingress-класса для обслуживания Ingress NGINX Controller.

Позволяет создать несколько контроллеров для обслуживания одного Ingress-класса.

i Если указать значение `nginx`, дополнительно будут обрабатываться Ingress-ресурсы без аннотации `kubernetes.io/ingress.class` или поля `spec.ingressClassName`.

По умолчанию: `nginx`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
ingressClass: nginx
```

- **spec.inlet**

строка

Обязательный параметр

Способ приема трафика из внешней сети. Изменение установленного способа невозможно.

Возможные значения:

- `LoadBalancer` — устанавливается Ingress-контроллер, и заказывается сервис с типом `LoadBalancer`;
- `LoadBalancerWithProxyProtocol` — устанавливается Ingress-контроллер, и заказывается сервис с типом `LoadBalancer`. Ingress-контроллер использует `proxy-protocol` для получения настоящего IP-адреса клиента;

i **Важно.** HTTP/3 (`enableHTTP3`) нельзя включать с этим инлетом, так как QUIC несовместим с PROXY protocol.

- `LoadBalancerWithSSLPassthrough` — устанавливается Ingress-контроллер, и заказывается сервис с типом `LoadBalancer`. Этот инлет включает функцию SSL Passthrough, которая позволяет настроить бэкенды на приём SSL-трафика напрямую, без терминации на Ingress-контроллере.

i Функция SSL Passthrough реализована путем перехвата всего трафика на настроенном HTTPS-порту (по умолчанию: `443`) и передачи его локальному TCP-прокси. Это полностью обходит NGINX и приводит к существенному снижению производительности.

Протокол SSL Passthrough использует Server Name Indication (SNI) и считывает данные о виртуальном домене из протокола согласования TLS, для чего требуется подходящий клиент.

Если имя хоста не совпадает с запрошенным именем хоста, запрос передается в NGINX на настроенный SSL Passthrough прокси-порт (по умолчанию: `442`), который затем передает запрос серверу по умолчанию.

i В отличие от HTTP-бэкендов, трафик для SSL Passthrough бэкендов отправляется на ClusterIP, а не на отдельные эндпоинты.

Поскольку SSL Passthrough работает на 4 уровне модели OSI (TCP), а не на уровне 7 (HTTP), использование SSL Passthrough делает невозможным использование всех остальных аннотаций, установленных для Ingress.

- `HostPort` — устанавливается Ingress-контроллер, который доступен на портах узлов через `hostPort`;
- `HostPortWithProxyProtocol` — устанавливается Ingress-контроллер, который доступен на портах узлов через `hostPort` и использует `proxy-protocol` для получения настоящего IP-адреса клиента.

i **Важно.** Перед использованием этого инлета убедитесь, что запросы к Ingress будут направляться только от доверенных источников. Для настройки ограничений воспользуйтесь параметром `acceptRequestsFrom`.

Важно. HTTP/3 (`enableHTTP3`) нельзя включать с этим инлетом, так как QUIC несовместим с PROXY protocol.

- `HostPortWithSSLPassthrough` — устанавливается Ingress-контроллер, который доступен на портах узлов через `hostPort`. Этот инлет включает функцию SSL Passthrough, которая позволяет настроить бэкенды на приём SSL-трафика напрямую, без терминации на Ingress-контроллере.

Укажите параметры работы инлета в секции `spec.HostPortWithSSLPassthrough`.

Протокол SSL Passthrough использует SNI и считывает данные о виртуальном домене из протокола согласования TLS, для чего требуется подходящий клиент.

Если имя хоста не совпадает с запрошенным именем хоста, запрос передается в NGINX на настроенный SSL Passthrough прокси-порт (по умолчанию: 442), который затем передает запрос серверу по умолчанию.

i Функция SSL Passthrough реализована путем перехвата всего трафика на настроенном HTTPS-порту (по умолчанию: 443) и передачи его локальному TCP-прокси. Это полностью обходит NGINX и приводит к существенному снижению производительности.

В отличие от HTTP-бэкендов, трафик для SSL Passthrough бэкендов отправляется на ClusterIP, а не на отдельные эндпоинты.

Поскольку SSL Passthrough работает на 4 уровне модели OSI (TCP), а не на уровне 7 (HTTP), использование SSL Passthrough делает невозможным использование всех остальных аннотаций, установленных для Ingress.

- `HostWithFailover` — устанавливаются два Ingress-контроллера — основной и резервный. Основной контроллер запускается в `hostNetwork`. Если поды основного контроллера недоступны, трафик уходит в резервный контроллер.

i На одном хосте может быть только один контроллер с данным типом инлета.

Убедитесь, что на узле свободны следующие порты: 80, 81, 443, 444, 4207, 4208.

Чтобы поменять инлет, удалите правила iptables и перезапустите поды kube-proxy или перезагрузите узлы, на которых размещались Ingress-контроллеры.

Данный инлет не может быть использован, если параметр `enableIstioSidecar` включен.

Замечание. При включении HTTP/3 (`enableHTTP3`) он применяется только к основному контроллеру; резервный контроллер всегда работает без HTTP/3.

Допустимые значения: `LoadBalancer`, `LoadBalancerWithSSLPassthrough`, `LoadBalancerWithProxyProtocol`, `HostPort`, `HostPortWithSSLPassthrough`, `HostPortWithProxyProtocol`, `HostWithFailover`

◦ `spec.legacySSL` булевый

Определяет, разрешены ли устаревшие версии протокола TLS и алгоритмов шифрования (cipher suites).

Работа разрешена для следующих версий протокола TLS: `TLSv1`, `TLSv1.1`, `TLSv1.2`, `TLSv1.3`.

Допустимые комбинации алгоритмов шифрования, расположенные в порядке от наиболее надёжной к наименее надёжной: `ECDHE-ECDSA-AES256-GCM-SHA384`, `ECDHE-RSA-AES256-GCM-SHA384`, `ECDHE-ECDSA-CHACHA20-POLY1305`, `ECDHE-RSA-CHACHA20-POLY1305`, `ECDHE-ECDSA-AES128-GCM-SHA256`, `ECDHE-RSA-AES128-GCM-SHA256`, `ECDHE-ECDSA-AES256-SHA384`, `ECDHE-RSA-AES256-SHA384`, `ECDHE-ECDSA-AES128-SHA256`, `ECDHE-RSA-AES128-SHA256`, `ECDHE-ECDSA-AES256-SHA`, `ECDHE-RSA-AES256-SHA`, `ECDHE-ECDSA-AES128-SHA`, `ECDHE-RSA-AES128-SHA`, `AES256-GCM-SHA384`, `AES128-GCM-SHA256`, `AES256-SHA256`, `AES128-SHA256`, `AES256-SHA`, `AES128-SHA`.

i По умолчанию включены только TLSv1.2, TLSv1.3 и самые новые алгоритмы шифрования.

◦ **spec.loadBalancer**
объект

Необязательный параметр.

Секция настроек инлета `LoadBalancer`.

▪ **spec.loadBalancer.acceptClientIPHeadersFrom**
массив строк

Определяет список доверенных подсетей в формате CIDR, запросы от которых содержат корректные заголовки `X-Forwarded-*/ProxyProtocol`.

По умолчанию: `["0.0.0.0/0"]`

Пример:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

▪ **Элемент массива**
строка

Шаблон: `^((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])|(\/(3[0-2]|[1-2][0-9]|0-9)))$`

▪ **spec.loadBalancer.annotations**
объект

Аннотации, которые будут назначены сервису для гибкой настройки балансировщика.

i Модуль не учитывает особенности указания аннотаций в различных облаках.

Если аннотации для заказа балансировщика применяются только при создании сервиса, для обновления подобных параметров пересоздайте ресурс `IngressNginxController` (или создайте новый и удалите старый).

▪ **spec.loadBalancer.behindL7Proxy**
булевый

Включает обработку и передачу заголовков `X-Forwarded-*`.

i **Внимание.** Перед использованием этой опции убедитесь, что запросы к Ingress направляются только от доверенных источников.

▪ **spec.loadBalancer.httpPort**
целочисленный

Внешний порт для небезопасного подключения по HTTP на балансировщике, создаваемом через сервис типа `LoadBalancer`.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты 80 и 443. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i **Внимание.** Поведение при смене портов зависит от облачного провайдера и может приводить к специфичным для каждого провайдера побочным эффектам (например, к пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. ACME HTTP-01 (например, Let's Encrypt через `cert-manager`) всегда проверяет доступность домена по TCP-порту 80. Если задать `httpsPort` без `httpPort` или изменить `httpPort` на значение отличное от 80, выпуск/продление сертификатов может перестать работать, если не используется DNS-01 или не настроен проброс порта 80 до контроллера на уровне внешнего балансировщика.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумевая стандартные порты 80/443). Если Ingress-контроллер доступен снаружи на нестандартных портах, редиректы могут вести на неправильный порт, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: 80

Допустимые значения: `1 <= X <= 65535`

■ **spec.loadBalancer.httpsPort**
целочисленный

Внешний порт для безопасного подключения по HTTPS на балансировщике, создаваемом через сервис типа `LoadBalancer`.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты `80` и `443`. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i

Внимание. Поведение при смене портов зависит от облачного провайдера и может приводить к специфичным для каждого провайдера побочным эффектам (например, к пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумевая стандартные порты `80/443`). При нестандартном `httpsPort` пользователи могут попадать на `443/tcr`, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: `443`

Допустимые значения: `1 <= X <= 65535`

■ **spec.loadBalancer.loadBalancerClass**
строка

Класс балансировщика входящих сетевых запросов (пробрасывается в параметр `spec.loadBalancerClass` заказанного сервиса с типом `LoadBalancer`).

■ **spec.loadBalancer.realIPHeader**
строка

Заголовок, из которого будет получен настоящий IP-адрес клиента.

i

Работает только при включении `behindL7Proxy`.

По умолчанию: `X-Forwarded-For`

Пример:

```
realIPHeader: CF-Connecting-IP
```

■ **spec.loadBalancer.sourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешен доступ к балансировщику.

i

Внимание. Облачный провайдер может не поддерживать данную опцию и игнорировать ее.

- Провайдеры поддерживающие опцию: AWS, Azure, GCP.
- Провайдеры игнорирующие опцию: Yandex Cloud.

Для остальных облачных провайдеров поведение может зависеть от особенностей реализации облака. Протестируйте работу опции перед использованием в production-среде.

■ **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0[0-9]))$`

○ **spec.loadBalancerWithProxyProtocol**
объект

Необязательный параметр.

Секция настроек инлета `LoadBalancerWithProxyProtocol`.

■ **spec.loadBalancerWithProxyProtocol.acceptClientIPHeadersFrom**
массив строк

Определяет список доверенных подсетей в формате CIDR, запросы от которых содержат корректные заголовки `X-Forwarded-*/ProxyProtocol`.

По умолчанию: `["0.0.0.0/0"]`

Пример:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

■ **Элемент массива**
строка

Шаблон: `^((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])|(\3[0-2]|\1[0-1][0-9]|0-9)))$`

■ **spec.loadBalancerWithProxyProtocol.annotations**
объект

Аннотации, которые будут назначены сервису для гибкой настройки балансировщика.

i

Модуль не учитывает особенности указания аннотаций в различных облаках.

Если аннотации для заказа балансировщика применяются только при создании сервиса, для обновления подобных параметров пересоздайте ресурс IngressNginxController (или создайте новый и удалите старый).

■ **spec.loadBalancerWithProxyProtocol.httpPort**
целочисленный

Внешний порт для небезопасного подключения по HTTP на балансировщике, создаваемом через сервис типа LoadBalancer.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты 80 и 443. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i

Внимание. Поведение при смене портов зависит от облачного провайдера и может приводить к специфичным для каждого провайдера побочным эффектам (например, к пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. ACME HTTP-01 (например, Let's Encrypt через cert-manager) всегда проверяет доступность домена по TCP-порту 80. Если задать `httpsPort` без `httpPort` или изменить `httpPort` на значение отличное от 80, выпуск/продление сертификатов может перестать работать, если не используется DNS-01 или не настроен проброс порта 80 до контроллера на уровне внешнего балансировщика.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумеваются стандартные порты 80/443). Если Ingress-контроллер доступен снаружи на нестандартных портах, редиректы могут вести на неправильный порт, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: 80

Допустимые значения: `1 <= X <= 65535`

■ **spec.loadBalancerWithProxyProtocol.httpsPort**
целочисленный

Внешний порт для безопасного подключения по HTTPS на балансировщике, создаваемом через сервис типа LoadBalancer.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты 80 и 443. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i

Внимание. Поведение при смене портов зависит от облачного провайдера и может приводить к специфичным для каждого провайдера побочным эффектам (например, к пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумеваются стандартные порты 80/443). При нестандартном `httpsPort` пользователи могут попадать на 443/tcr, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: 443

Допустимые значения: `1 <= X <= 65535`

■ **spec.loadBalancerWithProxyProtocol.loadBalancerClass**
строка

Класс балансировщика входящих сетевых запросов (пробрасывается в параметр `spec.loadBalancerClass` заказанного сервиса с типом `LoadBalancer`).

■ **spec.loadBalancerWithProxyProtocol.sourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешен доступ к балансировщику.

i **Внимание.** Облачный провайдер может не поддерживать данную опцию и игнорировать ее. Провайдеры поддерживающие опцию: AWS, Azure, GCP. Провайдеры игнорирующие опцию: Yandex Cloud.

Для остальных облачных провайдеров поведение может зависеть от особенностей реализации облака. Протестируйте работу опции перед использованием в production-среде.

■ **Элемент массива**
строка

Шаблон: `^((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])|(\3[0-2]|\1[0-1][0-9]|0[0-9]))$`

○ **spec.loadBalancerWithSSLPassthrough**
объект

Необязательный параметр.

Секция настроек инлета `LoadBalancerWithSSLPassthrough`.

■ **spec.loadBalancerWithSSLPassthrough.annotations**
объект

Аннотации, которые будут назначены сервису для гибкой настройки балансировщика.

i Модуль не учитывает особенности указания аннотаций в различных облаках.

Если аннотации для заказа балансировщика применяются только при создании сервиса, для обновления подобных параметров пересоздайте ресурс `IngressNginxController` (или создайте новый и удалите старый).

■ **spec.loadBalancerWithSSLPassthrough.httpPort**
целочисленный

Внешний порт для небезопасного подключения по HTTP на балансировщике, создаваемом через сервис типа `LoadBalancer`.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты `80` и `443`. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i **Внимание.** Поведение при смене портов зависит от облачного провайдера и может приводить к провайдер-специфичным побочным эффектам (например, пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. ACME HTTP-01 (например, Let's Encrypt через `cert-manager`) всегда проверяет доступность домена по TCP-порту `80`. Если задать `httpsPort` без `httpPort` или изменить `httpPort` на значение отличное от `80`, выпуск/продление сертификатов может перестать работать, если не используется DNS-01 или не настроен проброс порта `80` до контроллера на уровне внешнего балансировщика.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумеваются стандартные порты `80/443`). Если Ingress-контроллер доступен снаружи на нестандартных портах, редиректы могут вести на неправильный порт, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: `80`

Допустимые значения: `1 <= X <= 65535`

■ **spec.loadBalancerWithSSLPassthrough.httpsPort**
целочисленный

Внешний порт для безопасного подключения по HTTPS на балансировщике, создаваемом через

сервис типа `LoadBalancer`.

Если не заданы оба параметра `httpPort` и `httpsPort`, сервис будет слушать стандартные порты 80 и 443. Если задан хотя бы один из этих параметров, для соответствующего типа подключения сервис будет слушать только явно указанные порты. Для типа подключения, для которого не задан параметр, ничего не меняется: прослушивается стандартный порт.

i

Внимание. Поведение при смене портов зависит от облачного провайдера и может приводить к провайдер-специфичным побочным эффектам (например, пересозданию балансировщика или смене его публичного IP-адреса). Эти эффекты могут быть не документированы; перед использованием в production-среде протестируйте обновление сервиса.

Внимание. Многие приложения (включая некоторые компоненты модулей Deckhouse) формируют абсолютные ссылки и редиректы без явного указания порта (подразумевая стандартные порты 80/443). При нестандартном `httpsPort` пользователи могут попадать на 443/tcp, а ссылки могут формироваться некорректно; рекомендуется использовать стандартные порты снаружи или настраивать внешний проброс/проксирование портов.

По умолчанию: 443

Допустимые значения: $1 \leq X \leq 65535$

■ **spec.loadBalancerWithSSLPassthrough.loadBalancerClass**
строка

Класс балансировщика входящих сетевых запросов (пробрасывается в параметр `spec.loadBalancerClass` заказанного сервиса с типом `LoadBalancer`).

■ **spec.loadBalancerWithSSLPassthrough.sourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешен доступ к балансировщику.

i

Внимание. Облачный провайдер может не поддерживать данную опцию и игнорировать ее.

Провайдеры поддерживающие опцию: AWS, Azure, GCP. Провайдеры игнорирующие опцию: Yandex Cloud.

Для остальных облачных провайдеров поведение может зависеть от особенностей реализации облака. Протестируйте работу опции перед использованием в production-среде.

■ **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

○ **spec.maxReplicas**
целочисленный

Максимальное количество реплик `LoadBalancer`, `LoadBalancerWithProxyProtocol` и `LoadBalancerWithSSLPassthrough` для HPA.

По умолчанию: 1

Допустимые значения: $1 \leq X$

○ **spec.minReplicas**
целочисленный

Минимальное количество реплик `LoadBalancer`, `LoadBalancerWithProxyProtocol` и `LoadBalancerWithSSLPassthrough` для HPA.

По умолчанию: 1

Допустимые значения: $1 \leq X$

○ **spec.nginxProfilingEnabled**
булевый

Включает профилирование NGINX с помощью Valgrind. Отчет об анализе памяти сохраняется в каталоге `/var/log/valgrind` на хосте. При включении Valgrind создает файл с именем `memcheck.${timestamp}.log`.

После завершения отладки не забудьте отключить режим профилирования!

Отчет содержит информацию о поведении процесса, включая возможные утечки памяти.

i

Внимание. Включение этой опции значительно увеличивает использование памяти и замедляет обработку запросов HTTP/HTTPS процессом NGINX.



Внимание. Когда включена опция, под контроллера ingress-nginx запускается в привилегированном режиме.

◦ **spec.nodeSelector**
объект

Аналогично параметру `spec.nodeSelector` у подов.

Если значение не указано или указано `false`, Deckhouse попытается определить значение автоматически.

Используйте формат стандартного списка `nodeSelector`. Поды инстанса унаследуют это поле как есть.

◦ **spec.resourcesRequests**
объект

Настройки максимальных значений CPU и памяти (memory), которые может запросить под при выборе узла. Если VPA выключен, максимальные значения становятся значениями по умолчанию.

▪ **spec.resourcesRequests.mode**
строка

Обязательный параметр

Режим управления запросами ресурсов.

По умолчанию: `VPA`

Допустимые значения: `VPA`, `Static`

▪ **spec.resourcesRequests.static**
объект

Настройки статического режима управления.

▪ **spec.resourcesRequests.static.cpu**
строка

Значение для запроса к CPU.

По умолчанию: `350m`

▪ **spec.resourcesRequests.static.memory**
строка

Значение для запроса к памяти.

По умолчанию: `500Mi`

▪ **spec.resourcesRequests.vpa**
объект

Настройки режима управления Vertical Pod Autoscaler (VPA).

▪ **spec.resourcesRequests.vpa.cpu**
объект

Настройки ограничений запросов к CPU.

▪ **spec.resourcesRequests.vpa.cpu.max**
строка

Максимальное значение запроса к CPU, которое может выставить VPA.

По умолчанию: `50m`

▪ **spec.resourcesRequests.vpa.cpu.min**
строка

Минимальное значение запроса к CPU, которое может выставить VPA.

По умолчанию: `10m`

▪ **spec.resourcesRequests.vpa.memory**
объект

Настройки ограничений запросов к памяти.

▪ **spec.resourcesRequests.vpa.memory.max**
строка

Максимальное значение запроса к памяти, которое может выставить VPA.

По умолчанию: `200Mi`

▪ **spec.resourcesRequests.vpa.memory.min**
строка

Минимальное значение запроса к памяти, которое может выставить VPA.

По умолчанию: `50Mi`

- **spec.resourcesRequests.vpa.mode**
строка

Режим работы VPA.

По умолчанию: `Initial`

Допустимые значения: `Initial`, `Auto`

- **spec.tolerations**
массив объектов

Аналогично параметру `spec.tolerations` у подов.

Если значение не указано или указано `false`, подам модуля автоматически устанавливаются все возможные `tolerations`.

Используйте формат стандартного списка `tolerations`. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**
строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**
строка

- **spec.tolerations.operator**
строка

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
целочисленный

- **spec.tolerations.value**
строка

- **spec.underscoresInHeaders**
булевый

Разрешает использовать символ нижнего подчеркивания в заголовках.

Вспомогательные ресурсы:

- об использовании нижнего подчеркивания в заголовках;
- почему нужно внимательно включать разрешение для нижних подчеркиваний в заголовках.

По умолчанию: `false`

- **spec.validationEnabled**
булевый

Включает валидацию Ingress-правил.

i **Внимание.** Включение валидации увеличивает нагрузку на master-узлы кластера.

По умолчанию: `true`

- **spec.waitLoadBalancerOnTerminating**
целочисленный

Количество секунд до того момента, когда эндпоинт `/healthz` начнёт возвращать код `500` после перехода пода в статус `Terminating`.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

- **spec**
объект

Обязательный параметр

- **spec.acceptRequestsFrom**
массив строк

Список адресов в формате CIDR, которым разрешено подключаться к контроллеру.

Независимо от инлета всегда проверяется непосредственный адрес (в логах содержится в поле `original_address`), с которого производится подключение, а не «адрес клиента», который может передаваться в некоторых инлетах через заголовки или с использованием `proxy protocol`.

Параметр реализован с помощью `map module`, и если адрес, с которого непосредственно производится подключение, не разрешен – NGINX закрывает соединение (используя `return 444`).

По умолчанию к контроллеру можно подключаться с любых адресов.

- **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0[0-9]))$`

- **spec.additionalHeaders**
объект

Дополнительные header'ы, которые будут добавлены к каждому запросу. Указываются в формате `ключ: значение(строка)`.

- **spec.additionalLogFields**
объект

Дополнительные поля, которые будут добавлены в логи nginx. Указываются в формате `ключ: значение(строка)`.

- **spec.annotationValidationEnabled**
булевый

Включить валидацию аннотаций Ingress-правил.

По умолчанию: `false`

- **spec.chaosMonkey**
булевый

Инструмент, позволяющий систематически вызывать случайные прерывания работы подов контроллера.

Предназначен для проверки Ingress-контроллера на реальную работу отказоустойчивости.

По умолчанию: `false`

- **spec.config**
объект

Секция настроек Ingress-контроллера, в которую в формате `ключ: значение(строка)` можно записать любые возможные опции.



Внимание. Ошибка в указании опций может привести к отказу в работе Ingress-контроллера.



Внимание. Не рекомендуется использовать данную опцию, так как не гарантируется обратная совместимость или работоспособность Ingress-контроллера.

- **spec.controllerVersion**
строка

Версия Ingress NGINX Controller.

По умолчанию используется версия из настроек модуля. Поддерживаемые версии: `1.10`, `1.12`, `1.14`.

Допустимые значения: `1.10`, `1.12`, `1.14`

- **spec.customErrors**
объект

Секция с настройкой кастомизации HTTP-ошибок.

Если секция определена, все параметры в ней являются обязательными, изменение любого параметра **приводит к перезапуску всех Ingress-контроллеров**.

- **spec.customErrors.codes**
массив строк

Обязательный параметр

Список кодов ответа (массив), при которых запрос будет перенаправляться на custom default backend.

- **Элемент массива**
строка

Шаблон: `^[1-5][0-9][0-9]$`

- **spec.customErrors.namespace**
строка

Обязательный параметр

Имя namespace, в котором будет находиться сервис, используемый как custom default backend.

Пример:

```
namespace: default
```

- **spec.customErrors.serviceName**
строка

Обязательный параметр

Имя сервиса, который будет использоваться как custom default backend.

Пример:

```
serviceName: custom-errors-backend-service
```

◦ **spec.defaultSSLCertificate**
объект

Сертификат, который используется:

- при запросах на `catch-all`-сервер (подразумевается директива `server nginx`). На `catch-all`-сервер попадают запросы, для которых нет соответствующего Ingress-ресурса;
- для Ingress-ресурсов, в которых не задан `secretName` в секции `tls`.

По умолчанию используется самоподписанный сертификат.

i **Внимание.** Параметр не влияет на сертификаты, используемые в Ingress-ресурсах модулей Deckhouse. Для указания сертификата, который будет использоваться в Ingress-ресурсах модулей Deckhouse, используйте глобальный параметр `modules.https.customCertificate`.

■ **spec.defaultSSLCertificate.secretRef**
объект

Ссылка на Secret для передачи Ingress-контроллеру.

■ **spec.defaultSSLCertificate.secretRef.name**
строка

Имя Secret'а, содержащего SSL-сертификат.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ **spec.defaultSSLCertificate.secretRef.namespace**
строка

Имя namespace, в котором находится Secret с SSL-сертификатом.

По умолчанию: `d8-ingress-nginx`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

◦ **spec.disableHTTP2**
булевый

Выключить ли HTTP/2.

По умолчанию: `false`

◦ **spec.enableIstioSidecar**
булевый

Добавить к подам контроллера аннотации для автоматического инжекта сайдбаров Istio. При включении этого параметра к подам Ingress-контроллера добавляются аннотации `sidecar.istio.io/inject: "true"` и `traffic.sidecar.istio.io/includeOutboundIPRanges: "<Service CIDR>"`. При создании таких подов к ним автоматически будут добавлены сайдбары Istio с помощью mutating webhook. После этого весь трафик в сторону Service CIDR будет перехватываться сайдбаром.

Чтобы воспользоваться этой функцией, необходимо доработать прикладные Ingress-ресурсы, добавив аннотации:

- `nginx.ingress.kubernetes.io/service-upstream: "true"` — с этой аннотацией Ingress-контроллер будет отправлять запросы на ClusterIP сервиса (из диапазона Service CIDR) вместо того, чтобы слать их напрямую в поды приложения. Сайдбар istio-proxu перехватывает трафик только в сторону диапазона Service CIDR, остальные запросы отправляются напрямую;
- `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — с данной аннотацией сайдбар сможет идентифицировать прикладной сервис, для которого предназначен запрос.

i **Внимание.** Данный параметр не может быть включен, если инлет Ingress-контроллера установлен в значение `HostWithFailover`.

◦ **spec.geoIP2**
объект

Опции для включения GeoIP2.

■ **spec.geoIP2.maxmindEditionIDs**
массив строк

Список ревизий баз данных, которые будут скачаны при старте.

Подробнее...

По умолчанию: `["GeoLite2-City", "GeoLite2-ASN"]`

- **Элемент массива**
строка

Допустимые значения: `GeoIP2-Anonymous-IP`, `GeoIP2-Country`, `GeoIP2-City`, `GeoIP2-Connection-Type`, `GeoIP2-Domain`, `GeoIP2-ISP`, `GeoIP2-ASN`, `GeoLite2-ASN`, `GeoLite2-Country`, `GeoLite2-City`

- **spec.geoIP2.maxmindLicenseKey**
строка

Лицензионный ключ для скачивания базы данных GeoIP2.

Указание ключа в конфигурации включает скачивание базы GeoIP2 при каждом старте контроллера. Подробнее о получении ключа.

- **spec.hostPort**
объект

Секция настроек для инлета `HostPort`.

- **spec.hostPort.behindL7Proxy**
булевый

Включает обработку и передачу заголовков `X-Forwarded-*`.



Внимание. При использовании этой опции вы должны быть уверены, что запросы к Ingress-контроллеру направляются только от доверенных источников. Одним из способов настройки ограничения может служить опция `acceptRequestsFrom`.

- **spec.hostPort.httpPort**
целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, возможность подключения по HTTP отсутствует.

Обязательный параметр, если не указан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPort.httpsPort**
целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, возможность подключения по HTTPS отсутствует.

Обязательный параметр, если не указан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.hostPort.realIPHeader**
строка

Заголовок, из которого будет получен настоящий IP-адрес клиента.

Работает **только** при включении `behindL7Proxy`.

По умолчанию: `X-Forwarded-For`

Пример:

```
realIPHeader: CF-Connecting-IP
```

- **spec.hostPortWithProxyProtocol**
объект

Секция настроек для инлета `HostPortWithProxyProtocol`.

- **spec.hostPortWithProxyProtocol.httpPort**
целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, возможность подключения по HTTP отсутствует.

Обязательный параметр, если не указан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPortWithProxyProtocol.httpsPort**

целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, возможность подключения по HTTPS отсутствует.

Обязательный параметр, если не указан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.hsts**

булевый

Включен ли HSTS. Подробнее....

По умолчанию: `false`

- **spec.hstsOptions**

объект

Параметры HTTP Strict Transport Security.

- **spec.hstsOptions.includeSubDomains**

булевый

Применять ли настройки HSTS ко всем поддоменам сайта.

По умолчанию: `false`

- **spec.hstsOptions.maxAge**

строка

Время в секундах, в течение которого браузер должен помнить, что сайт доступен только с помощью HTTPS.

По умолчанию: `31536000`Шаблон: `^[1-9][0-9]*$`

Пример:

```
maxAge: '31536000'
```

- **spec.hstsOptions.preload**

булевый

Добавлять ли сайт в список предзагрузки.

Эти списки используются современными браузерами и разрешают подключение к сайту только по HTTPS.

По умолчанию: `false`

- **spec.ingressClass**

строка

Имя Ingress-класса для обслуживания Ingress NGINX Controller.

Позволяет создать несколько контроллеров для обслуживания одного Ingress-класса.

Важно. Если указать значение “nginx”, дополнительно будут обрабатываться Ingress-ресурсы без аннотации `kubernetes.io/ingress.class` или поля `spec.ingressClassName`.По умолчанию: `nginx`Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
ingressClass: nginx
```

- **spec.inlet**

строка

Обязательный параметр

Способ приема трафика из внешней сети. Изменение установленного способа невозможно.

- `LoadBalancer` — устанавливается Ingress-контроллер и заказывается сервис с типом `LoadBalancer`;
- `LoadBalancerWithProxyProtocol` — устанавливается Ingress-контроллер и заказывается сервис с типом `LoadBalancer`. Ingress-контроллер использует `proxy-protocol` для получения настоящего IP-адреса клиента;

- `HostPort` — устанавливается Ingress-контроллер, который доступен на портах узлов через `hostPort`;

Требуется обязательная установка параметров в секции `spec.HostPort`.

- `HostPortWithProxyProtocol` — устанавливается Ingress-контроллер, который доступен на портах узлов через `hostPort` и использует `proxy-protocol` для получения настоящего IP-адреса клиента;

Требуется обязательная установка параметров в секции `spec.HostPortWithProxyProtocol`.

Внимание. При использовании этого инлета, убедитесь, что запросы к Ingress-контроллеру направляются только от доверенных источников. Одним из способов настройки ограничения может служить параметр `acceptRequestsFrom`.

- `HostWithFailover` — устанавливаются два Ingress-контроллера — основной и резервный. Основной контроллер запускается в `hostNetwork`. Если поды основного контроллера недоступны, трафик уходит в резервный контроллер;

Внимание. На одном хосте может быть только один контроллер с данным типом инлета.

Внимание. Необходимо, чтобы на узле были свободны следующие порты: 80, 81, 443, 444, 4207, 4208.

Внимание. Чтобы поменять инлет, удалите правила iptables и перезапустите поды `kube-проху` или перезагрузите узлы, на которых размещались Ingress-контроллеры.

Внимание. Данный инлет не может быть использован, если параметр `enableIstioSidecar` включен.

Допустимые значения: `LoadBalancer`, `LoadBalancerWithProxyProtocol`, `HostPort`, `HostPortWithProxyProtocol`, `HostWithFailover`

◦ `spec.legacySSL` булевый

Определяет, разрешены ли устаревшие версии протокола TLS и алгоритмов шифрования (cipher suites).

Работа разрешена для следующих версий протокола TLS: `TLSv1`, `TLSv1.1`, `TLSv1.2`, `TLSv1.3`.

Допустимые комбинации алгоритмов шифрования, расположенные в порядке от наиболее надёжной к наименее надёжной: `ECDSA-ECDHE-AES256-GCM-SHA384`, `ECDSA-ECDHE-RSA-AES256-GCM-SHA384`, `ECDSA-ECDHE-CHACHA20-POLY1305`, `ECDSA-ECDHE-RSA-CHACHA20-POLY1305`, `ECDSA-ECDHE-AES128-GCM-SHA256`, `ECDSA-ECDHE-RSA-AES128-GCM-SHA256`, `ECDSA-ECDHE-AES256-SHA384`, `ECDSA-ECDHE-RSA-AES256-SHA384`, `ECDSA-ECDHE-AES128-SHA256`, `ECDSA-ECDHE-RSA-AES128-SHA256`, `ECDSA-ECDHE-AES256-SHA`, `ECDSA-ECDHE-RSA-AES256-SHA`, `ECDSA-ECDHE-AES128-SHA`, `ECDSA-ECDHE-RSA-AES128-SHA`, `AES256-GCM-SHA384`, `AES128-GCM-SHA256`, `AES256-SHA256`, `AES128-SHA256`, `AES256-SHA`, `AES128-SHA`.

По умолчанию включены только `TLSv1.2`, `TLSv1.3` и самые новые алгоритмы шифрования.

◦ `spec.loadBalancer` объект

Необязательный параметр.

Секция настроек для инлета `LoadBalancer`.

- `spec.loadBalancer.annotations`
объект

Аннотации, которые будут проставлены сервису для гибкой настройки балансировщика.

Внимание. Модуль не учитывает особенности указания аннотаций в различных облаках. Если аннотации для заказа `LoadBalancer` применяются только при создании сервиса, для обновления подобных параметров необходимо будет пересоздать `IngressNginxController` (или создать новый, затем удалив старый).

- `spec.loadBalancer.behindL7Proxy`
булевый

Включает обработку и передачу заголовков `X-Forwarded-*`.

Внимание. При использовании этой опции вы должны быть уверены, что запросы к Ingress направляются только от доверенных источников.

- `spec.loadBalancer.loadBalancerClass`
строка

Класс балансировщика входящих сетевых запросов (пробрасывается в параметр `spec.loadBalancerClass` заказанного сервиса с типом `LoadBalancer`).

- `spec.loadBalancer.realIPHeader`

строка

Заголовок, из которого будет получен настоящий IP-адрес клиента.

Работает только при включении `behindL7Proxy`.

По умолчанию: `X-Forwarded-For`

Пример:

```
realIPHeader: CF-Connecting-IP
```

- **spec.loadBalancer.sourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешен доступ на балансировщик.

i

Внимание. Облачный провайдер может не поддерживать данную опцию и игнорировать ее. Провайдеры поддерживающие опцию: *AWS*, *GCP*, *Azure* Провайдеры игнорирующие опцию: *YandexCloud* Для остальных облачных провайдеров поведение может зависеть от особенностей реализации облака. Перед промышленным использованием опции рекомендуется тестирование.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.loadBalancerWithProxyProtocol**

объект

Необязательный параметр.

Секция настроек для инлета `LoadBalancerWithProxyProtocol`.

- **spec.loadBalancerWithProxyProtocol.annotations**

объект

Аннотации, которые будут проставлены сервису для гибкой настройки балансировщика.

Внимание. Модуль не учитывает особенности указания аннотаций в различных облаках. Если аннотации для заказа `LoadBalancer`'а применяются только при создании сервиса, для обновления подобных параметров необходимо будет пересоздать `IngressNginxController` (или создать новый, затем удалив старый).

- **spec.loadBalancerWithProxyProtocol.loadBalancerClass**

строка

Класс балансировщика входящих сетевых запросов (пробрасывается в параметр `spec.loadBalancerClass` заказанного сервиса с типом `LoadBalancer`).

- **spec.loadBalancerWithProxyProtocol.sourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешен доступ на балансировщик.

i

Внимание. Облачный провайдер может не поддерживать данную опцию и игнорировать ее. Провайдеры поддерживающие опцию: *AWS*, *GCP*, *Azure* Провайдеры игнорирующие опцию: *YandexCloud* Для остальных облачных провайдеров поведение может зависеть от особенностей реализации облака. Перед промышленным использованием опции рекомендуется тестирование.

- **Элемент массива**

строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.maxReplicas**

целочисленный

Максимально количество реплик `LoadBalancer` и `LoadBalancerWithProxyProtocol` для HPA.

По умолчанию: `1`

Допустимые значения: `1 <= X`

- **spec.minReplicas**

целочисленный

Минимальное количество реплик `LoadBalancer` и `LoadBalancerWithProxyProtocol` для HPA.

По умолчанию: `1`

Допустимые значения: `1 <= X`

- **spec.nodeSelector**

объект

Как в `spec.nodeSelector` у подов.

Если значение не указано или указано `false`, будет использоваться автоматика.

Формат: стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- **spec.resourcesRequests**

объект

Настройки максимальных значений CPU и memory, которые может запросить под при выборе узла (если VPA выключен, максимальные значения становятся значениями по умолчанию).

- **spec.resourcesRequests.mode**

строка

Обязательный параметр

Режим управления реквестами ресурсов.

По умолчанию: `VPA`

Допустимые значения: `VPA`, `Static`

- **spec.resourcesRequests.static**

объект

Настройки статического режима управления.

- **spec.resourcesRequests.static.cpu**

строка

Значение для реквеста к CPU.

По умолчанию: `350m`

- **spec.resourcesRequests.static.memory**

строка

Значение для реквеста к memory.

По умолчанию: `500Mi`

- **spec.resourcesRequests.vpa**

объект

Настройки VPA режима управления.

- **spec.resourcesRequests.vpa.cpu**

объект

Настройки для CPU.

- **spec.resourcesRequests.vpa.cpu.max**

строка

Максимальное значение, которое может выставить VPA для реквеста к CPU.

По умолчанию: `50m`

- **spec.resourcesRequests.vpa.cpu.min**

строка

Минимальное значение, которое может выставить VPA для реквеста к CPU.

По умолчанию: `10m`

- **spec.resourcesRequests.vpa.memory**

объект

Значение для запроса memory.

- **spec.resourcesRequests.vpa.memory.max**

строка

Максимальное значение, которое может выставить VPA для реквеста к memory.

По умолчанию: `200Mi`

- **spec.resourcesRequests.vpa.memory.min**

строка

Минимальное значение, которое может выставить VPA для реквеста к memory.

По умолчанию: `50Mi`

- **spec.resourcesRequests.vpa.mode**

строка

Режим работы VPA.

По умолчанию: `Initial`

Допустимые значения: `Initial`, `Auto`

- **spec.tolerations**

массив объектов

Как в `spec.tolerations` у подов.

Если значение не указано или указано `false`, будет использоваться автоматика.

Формат: стандартный список toleration. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**

строка

- **spec.tolerations.operator**

строка

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**

целочисленный

- **spec.tolerations.value**

строка

- **spec.underscoresInHeaders**

булевый

Разрешены ли нижние подчеркивания в заголовках.

Подробнее....

Почему не стоит бездумно это включать.

По умолчанию: `false`

- **spec.validationEnabled**

булевый

Включить валидацию Ingress-правил.

i

Внимание. Включение валидации увеличивает нагрузку на master-узлы кластера.

По умолчанию: `true`

- **spec.waitLoadBalancerOnTerminating**

целочисленный

Количество секунд до того, как /healthz начнет возвращать код 500, когда под перейдет в статус Terminating.

FAQ

Как разрешить доступ к приложению внутри кластера только от Ingress-контроллера?

Если необходимо ограничить доступ к вашему приложению внутри кластера исключительно от подов Ingress-контроллера, необходимо в под с приложением добавить контейнер с kube-rbac-proxy, как показано в примере ниже:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: my-app
  replicas: 1
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-sa
      containers:
        - name: my-cool-app
          image: mycompany/my-app:v0.5.3
          args:
            - "--listen=127.0.0.1:8080"
          livenessProbe:
            httpGet:
              path: /healthz
              port: 443
              scheme: HTTPS
        - name: kube-rbac-proxy
          image: flant/kube-rbac-proxy:v0.1.0
          # Рекомендуется использовать прокси из репозитория Deckhouse.
          args:
            - "--secure-listen-address=0.0.0.0:443"
            - "--config-file=/etc/kube-rbac-proxy/config-file.yaml"
            - "--v=2"
            - "--logtostderr=true"
          # Если kube-apiserver недоступен, аутентификация и авторизация пользователей невозможна.
          # Stale Cache хранит результаты успешной авторизации и используется лишь в случае, если
          apiserver недоступен.
            - "--stale-cache-interval=1h30m"
          ports:
            - containerPort: 443
              name: https
          volumeMounts:
            - name: kube-rbac-proxy
              mountPath: /etc/kube-rbac-proxy
          volumes:
            - name: kube-rbac-proxy
              configMap:
                name: kube-rbac-proxy

```

Приложение принимает запросы на адресе `127.0.0.1`, это означает, что по незащищенному соединению к нему можно подключиться только внутри пода. Прокси прослушивает порт на адресе `0.0.0.0` и перехватывает весь внешний трафик к поду.

Как выдать минимальные права для ServiceAccount?

Чтобы аутентифицировать и авторизовывать пользователей с помощью kube-apiserver, у прокси должны быть права на создание `TokenReview` и `SubjectAccessReview`.

В кластерах DKP уже есть готовая ClusterRole — **d8-rbac-proxy**, создавать её самостоятельно не требуется! Свяжите её с ServiceAccount вашего Deployment'a, как показано в примере ниже.

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
  namespace: my-namespace
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-namespace:my-sa:d8-rbac-proxy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: d8:rbac-proxy
subjects:
- kind: ServiceAccount
  name: my-sa
  namespace: my-namespace

```

Конфигурация Kube-RBAC-Proxy

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-rbac-proxy
data:
  config-file.yaml: |+
    excludePaths:
      - /healthz
    # Не требуем авторизацию для liveness пробы.
    upstreams:
      - upstream: http://127.0.0.1:8081/
    # Адрес upstream-сервиса, на который будет перенаправлен входящий трафик.
    path: /
    # Путь, обрабатываемый прокси, по которому принимаются запросы и перенаправляются на upstream.
    authorization:
      resourceAttributes:
        namespace: my-namespace
        apiGroup: apps
        apiVersion: v1
        resource: deployments
        subresource: http
        name: my-app

```

Согласно конфигурации, у пользователя должны быть права доступа к Deployment с именем `my-app` и его дополнительному ресурсу `http` в пространстве имён `my-namespace`.

Выглядят такие права в виде RBAC следующим образом:

```

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kube-rbac-proxy:my-app
  namespace: my-namespace
rules:
- apiGroups: ["apps"]
  resources: ["deployments/http"]
  resourceNames: ["my-app"]
  verbs: ["get", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kube-rbac-proxy:my-app
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kube-rbac-proxy:my-app
subjects:
# Все пользовательские сертификаты ingress-контроллеров выписаны для одной конкретной группы.
- kind: Group
  name: ingress-nginx:auth

```

Для Ingress-ресурса добавьте параметры:

```

nginx.ingress.kubernetes.io/backend-protocol: HTTPS
nginx.ingress.kubernetes.io/configuration-snippet: |
  proxy_ssl_certificate /etc/nginx/ssl/client.crt;
  proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
  proxy_ssl_protocols TLSv1.2;
  proxy_ssl_session_reuse on;

```

Подробнее о том, как работает аутентификация по сертификатам, можно прочитать в документации Kubernetes.

Как сконфигурировать балансировщик нагрузки для проверки доступности IngressNginxController?

В ситуации, когда Ingress-контроллер размещен за балансировщиком нагрузки, рекомендуется сконфигурировать балансировщик для проверки доступности узлов Ingress-контроллер с помощью HTTP-запросов или TCP-подключений. В то время как тестирование с помощью TCP-подключений представляет собой простой и универсальный механизм проверки доступности, мы рекомендуем использовать проверку на основе HTTP-запросов со следующими параметрами:

- протокол: HTTP;
- путь: /healthz;
- порт: 80 (в случае использования инлета `HostPort` нужно указать номер порта, соответствующий параметру `httpPort`).

Как настроить работу через MetalLB с доступом только из внутренней сети?

Пример MetalLB с настройками доступа только из внутренней сети:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  loadBalancer:
    sourceRanges:
    - 192.168.0.0/24

```



Для работы необходимо включить параметр `svcSourceRangeCheck` в модуле `cni-cilium`.

Как добавить дополнительные поля для логирования в nginx-controller?

Пример добавления дополнительных полей:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  additionalLogFields:
    my-cookie: "$cookie_MY_COOKIE"

```

Как включить HorizontalPodAutoscaling для IngressNginxController?



Режим HPA возможен только для контроллеров с инлетом `LoadBalancer` или `LoadBalancerWithProxyProtocol`.

Режим HPA возможен только при `minReplicas != maxReplicas`, в противном случае deployment `hpa-scaler` не создается.

Для включения HPA используйте атрибуты `minReplicas` и `maxReplicas` в IngressNginxController CR.

IngressNginxController разворачивается с помощью DaemonSet. DaemonSet не предоставляет возможности горизонтального масштабирования, поэтому создается дополнительный deployment `hpa-scaler` и HPA resource, который следит за предварительно созданной метрикой `prometheus-metrics-adapter-d8-ingress-nginx-cpu-utilization-for-hpa`. Если CPU utilization превысит 50%, HPA закажет новую реплику для `hpa-scaler` (с учетом `minReplicas` и `maxReplicas`).

Deployment `hpa-scaler` обладает HardPodAntiAffinity (запрет на размещение подов с одинаковыми лейблами на одном узле), поэтому он попытается выделить для себя новый узел (если это возможно в рамках своей группы узлов), куда автоматически будет размещен еще один instance Ingress-контроллера.



- Минимальное реальное количество реплик IngressNginxController не может быть меньше минимального количества узлов в группе узлов, куда он разворачивается.
- Максимальное реальное количество реплик IngressNginxController не может быть больше максимального количества узлов в группе узлов, куда он разворачивается.

Как использовать IngressClass с установленными IngressClassParameters?

Начиная с версии 1.1 IngressNginxController, Deckhouse создает объект IngressClass самостоятельно. Если вы хотите использовать свой IngressClass с установленными IngressClassParameters, достаточно добавить к нему label `ingress-class.deckhouse.io/external: "true"`:

```

apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    ingress-class.deckhouse.io/external: "true"
  name: my-super-ingress
spec:
  controller: ingress-nginx.deckhouse.io/my-super-ingress
  parameters:
    apiGroup: elbv2.k8s.aws
    kind: IngressClassParams
    name: awesome-class-cfg

```

В этом случае, при указании данного IngressClass в CRD IngressNginxController, Deckhouse не будет создавать объект, а использует существующий.

Как отключить сборку детализированной статистики Ingress-ресурсов?

По умолчанию Deckhouse собирает подробную статистику со всех Ingress-ресурсов в кластере. Этот процесс может приводить к высокой нагрузке системы мониторинга.

Для отключения сбора статистики добавьте лейбл `ingress.deckhouse.io/discard-metrics: "true"` к соответствующему пространству имён или Ingress-ресурсу.

Пример отключения сбора статистики (метрик) для всех Ingress-ресурсов в пространстве имен `review-1`:

```
d8 k label ns review-1 ingress.deckhouse.io/discard-metrics=true
```

Пример отключения сбора статистики (метрик) для всех Ingress-ресурсов `test-site` в пространстве имен `development`:


```
d8 k label ingress test-site -n development ingress.deckhouse.io/discard-metrics=true
```

Как корректно вывести из эксплуатации (drain) узел с запущенным IngressNginxController?

Доступно два способа корректного вывода из эксплуатации узла, на котором запущен IngressNginxController.

1. С помощью аннотации.

Аннотация будет автоматически удалена после завершения операции.

```
d8 k annotate node <node_name> update.node.deckhouse.io/drainng=user
```

2. С помощью d8 k drain.

При использовании стандартной команды d8 k drain необходимо указать флаг `--force` даже при наличии `-ignore-daemonsets`, поскольку IngressNginxController развёрнут с использованием Advanced DaemonSet:

```
d8 k drain <node_name> --delete-emptydir-data --ignore-daemonsets --force
```

Как включить Web Application Firewall (WAF)?

Для защиты веб-приложений от L7-атак используется программное обеспечение известное как Web Application Firewall (WAF). В ingress-nginx контроллер встроено WAF под названием ModSecurity (проект Open Worldwide Application Security).

По умолчанию ModSecurity выключен.

Включение ModSecurity

Для включения ModSecurity необходимо задать параметры в кастомном ресурсе IngressNginxController, в секции config:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <имя_контроллера>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
```

После применения настроек ModSecurity начнет работать для всего трафика, проходящего через данный ingress-nginx контроллер. При этом используется режим аудита (`DetectionOnly`) и базовая рекомендуемая конфигурация.

Настройка ModSecurity

ModSecurity можно настраивать двумя способами:

1. Для всего ingress-nginx контроллера
 - необходимые директивы описываются в секции `config.modsecurity-snippet` в кастомном ресурсе IngressNginxController, как в примере выше.
2. Для каждого кастомного ресурса Ingress по отдельности
 - необходимые директивы описываются в аннотации `nginx.ingress.kubernetes.io/modsecurity-snippet: |` непосредственно в манифестах Ingress.

Чтобы включить выполнение правил (а не только логирование), добавьте директиву `SecRuleEngine On` по примеру ниже:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <имя_контроллера>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
```

Добавление SecRule

Пример правила для ограничения количества аргументов в URL-адресе запроса. Если количество аргументов

превышает 10, сервер отклоняет запрос с кодом ошибки 400 (Bad request).

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <имя_контролера>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
      SecRule &ARGS "@gt 10"
    "id:100100,phase:2,deny,status:400,log,auditlog,severity:WARNING,msg:\"too many args\""
```

Полный перечень и описание директив вы можете найти в официальной документации.

Включение OWASP Core Rule Set (CRS)

Для включения OWASP Core Rule Set (CRS) необходимо добавить `config.enable-owasp-modsecurity-crs` в кастомном ресурсе `IngressNginxController`, как показано на примере ниже.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <name_of_the_controller>
spec:
  config:
    enable-modsecurity: "true"
    enable-owasp-modsecurity-crs: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
```

Примеры

Пример для AWS (Network Load Balancer)

При создании балансировщика используются все доступные зоны в кластере.

В каждой зоне балансировщик получает публичный IP. Если в зоне есть instance с Ingress-контроллером, A-запись с IP-адресом балансировщика из этой зоны автоматически добавляется к доменному имени балансировщика.

Если в зоне не остается instance с Ingress-контроллером, тогда IP автоматически удаляется из DNS.

В том случае, если в зоне всего один инстанс с Ingress-контроллером, при перезапуске пода IP-адрес балансировщика этой зоны будет временно исключен из DNS.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

Пример для GCP / Yandex Cloud / Azure

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
```



В GCP на узлах должна присутствовать аннотация, разрешающая принимать подключения на внешние адреса для сервисов с типом NodePort.

Пример для OpenStack

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main-lbwpp
spec:
  inlet: LoadBalancerWithProxyProtocol
  ingressClass: nginx
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"

```

Пример для bare metal

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostWithFailover
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend

```

Пример для bare metal (при использовании внешнего балансировщика, например Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp и др.)

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true

```

Пример для bare metal (балансировщик MetalLB в режиме BGP LoadBalancer)

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend

```

В случае использования MetalLB его speaker-поды должны быть запущены на тех же узлах, что и поды Ingress-контроллера.

Контроллер должен получать реальные IP-адреса клиентов — поэтому его Service создается с параметром `externalTrafficPolicy: Local` (запрещающий межузловую SNAT), и для принятия данного параметра MetalLB speaker анонсирует этот Service только с тех узлов, в которых запущены целевые поды.

Для этого примера конфигурация модуля `metallb` должна быть следующей:

```
metallb:
  speaker:
    nodeSelector:
      node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

Пример для bare metal (балансировщик MetallB в режиме L2 LoadBalancer)

1. Включите модуль metallb:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

2. Создайте ресурс MetalLoadBalancerClass:

i Metallb-балансировщики должны размещаться на тех же узлах, что и ingress-контроллеры. В типовых сценариях развертывания для этого используются frontend-узлы (для развертывания ingress-контроллеров и Metallb-балансировщиков на frontend-узлах используйте в их манифестах аннотацию `node-role.deckhouse.io/frontend: ""`).

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  type: L2
```

3. Создайте ресурс IngressNginxController:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    loadBalancerClass: ingress
    annotations:
      # Количество адресов, которые будут выделены из пула, объявленного в
      MetalLoadBalancerClass.
      network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
      # Селектор и tolerations. Поды ingress-controller должны быть размещены на тех же узлах, что
      и поды MetallB speaker.
      nodeSelector:
        node-role.deckhouse.io/frontend: ""
      tolerations:
        - effect: NoExecute
          key: dedicated.deckhouse.io
          value: frontend
          operator: Equal
```

i При создании ingress-контроллера также можно указать определенные IP-адреса из пула, которые будут ему присвоены. Для указания адресов, которые должны быть присвоены сервису, используйте аннотацию `network.deckhouse.io/load-balancer-ips`. Если желаемых адресов больше одного, то также должна присутствовать аннотация `network.deckhouse.io/l2-load-balancer-external-ips-count`, в которой необходимо указать количество выделяемых адресов из пула (оно не должно быть меньше количества адресов, перечисленных в `network.deckhouse.io/load-balancer-ips`). Пример использования аннотаций для присвоения сервису определенных адресов из пула.

4. Платформа создаст сервис с типом `LoadBalancer`, которому будет присвоено заданное количество адресов:

```
$ d8 k -n d8-ingress-nginx get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
main-load-balancer                 LoadBalancer       10.222.130.11   192.168.2.100,192.168.2.101,192.168.2.102
80:30689/TCP,443:30668/TCP        11s
```

Пример разделения доступа между публичной и административной зонами

Во многих приложениях один и тот же бэкенд обслуживает как публичную часть, так и административный интерфейс. Например:

- `https://example.com` — публичная зона;
- `https://admin.example.com` — административная зона, к которой доступ должен быть ограничен (`ACL`, `mTLS`, `IP whitelist` и т.д.).

При таком сценарии рекомендуем выносить административный трафик в отдельный Ingress-контроллер (при необходимости с отдельным Ingress-классом) и ограничивать доступ к нему с помощью параметра `spec.acceptRequestsFrom`.

Особенности использования одного Ingress-контроллера

Рассмотрим пример, когда для обслуживания запросов из публичной зоны и административного интерфейса используется один Ingress-контроллер.

Пример конфигурации Ingress-ресурсов для такого случая:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: nginx # Ingress-ресурс для административного трафика связан с тем же Ingress-
  контроллером, что и Ingress-ресурс для публичного трафика.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: nginx # Ingress-ресурс для публичного трафика связан с тем же Ingress-
  контроллером, что и Ingress-ресурс для административного трафика.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80

```

При включенной обработке и передаче заголовков `X-Forwarded-*` бэкенд может опираться на заголовок `x-forwarded-host` при принятии решений об авторизации. И для приведенного выше примера возможна ситуация, когда через Ingress-ресурс для обслуживания публичного трафика можно подключиться к административной зоне, используя `x-forwarded-host`. Поэтому при использовании этой опции вы должны быть уверены, что запросы к Ingress-контроллеру направляются только от доверенных источников.

Использование отдельных Ingress-контроллеров

Чтобы избежать ситуации, описанной выше (когда при включенной обработке и передаче заголовков `X-Forwarded-*` можно, например, через Ingress-ресурс для обслуживания публичного трафика подключиться к административной зоне, используя `x-forwarded-host`), рекомендуем:

- настроить правила доступа на уровне Ingress-ресурсов,
- использовать разные Ingress-контроллеры,
- на Ingress-контроллерах ограничить, с каких адресов можно подключаться к ним.

Пример конфигурации Ingress-ресурсов для такого случая:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: admin-nginx # Ingress-ресурс для административного трафика связан с отдельным
  Ingress-контроллером.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: public-nginx # Ingress-ресурс для публичного трафика связан с отдельным
  Ingress-контроллером.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80

```

Пример Ingress-контроллера, который обслуживает административные Ingress-ресурсы и принимает подключения только из заданных подсетей:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: admin
spec:
  ingressClass: admin-nginx
  inlet: HostPort
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true

```

В этом примере:

- Ingress-контроллер доступен на портах узлов через инлет `HostPort`;
- Параметр `acceptRequestsFrom` разрешает подключение к контроллеру только из перечисленных подсетей;
- Даже если внешний балансировщик или клиент может передавать свои значения заголовков `X-Forwarded-*`, решение о допуске соединения до контроллера принимается по реальному адресу подключения, а не по заголовкам.
- Административные Ingress-ресурсы (в данном примере `admin-ingress`) обслуживаются этим контроллером согласно настроенному Ingress-классу.

Пример Ingress-контроллера, который обслуживает Ingress-ресурсы для публичного трафика:

```

apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: public
spec:
  ingressClass: public-nginx
  inlet: HostPort
  hostPort:
    httpPort: 8080
    httpsPort: 8443
    behindL7Proxy: true

```

Модуль istio

i Веб-интерфейсы, связанные с модулем: istio

Таблица совместимости поддерживаемых версий

Версия Istio	Версии K8S, поддерживаемые Istio	Статус в текущем релизе D8
1.25	1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35	Поддерживается
1.21	1.26, 1.27, 1.28, 1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35	Устарела и будет удалена

Задачи, которые решает Istio

Istio — фреймворк централизованного управления сетевым трафиком, реализующий подход Service Mesh.

Istio решает для приложений следующие задачи:

- Использование Mutual TLS.
- Авторизация доступа между сервисами.
- Маршрутизация запросов.
- Управление балансировкой запросов между эндпоинтами сервиса.
- Повышение Observability.
- Организация мульти-ЦОД кластера за счет объединения кластеров в единый Service Mesh (мультикластер).
- Объединение разрозненных кластеров в федерацию с возможностью предоставлять стандартный (в понимании Service Mesh) доступ к избранным сервисам.

i Рекомендуем ознакомиться с видео, где рассмотрена архитектура Istio и оценены накладные расходы.

Mutual TLS

Это основной метод взаимной аутентификации сервисов. Принцип основан на том, что для всех исходящих запросов проверяется сертификат сервера, а для входящих - клиентский сертификат. После проверки сайдкар-прокси получает возможность идентифицировать удаленный узел и использовать эти данные для аутентификации или для прикладных целей.

Каждый сервис получает собственный идентификатор в формате `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`, где `TrustDomain` в нашем случае — это домен кластера. Каждому сервису можно выделять собственный `ServiceAccount` или использовать стандартный «default». Полученный идентификатор сервиса можно использовать как в правилах авторизации, так и в прикладных целях. Именно этот идентификатор используется в качестве удостоверяемого имени в TLS-сертификатах.

Данные настройки можно переопределить на уровне неймспейса.

Авторизация

Ресурс `AuthorizationPolicy` отвечает за управление авторизацией. После создания этого ресурса для сервиса, для определения судьбы запроса, используется следующий алгоритм:

- Если запрос попадает под политику `DENY` — запретить запрос;
- Если для данного сервиса нет политик `ALLOW` — разрешить запрос;
- Если запрос попадает под политику `ALLOW` — разрешить запрос;
- Все остальные запросы — запретить.

Иными словами, если явно что-то запретить, работает только запрет. Если же что-то явно разрешить, будут разрешены только явно одобренные запросы (запреты при этом имеют приоритет).

Для написания правил авторизации доступны следующие аргументы:

- идентификаторы сервисов и wildcard на их основе (`mycluster.local/ns/myns/sa/myapp` или `mycluster.local/*`);
- неймспейсы;

- диапазоны IP;
- HTTP-заголовки;
- JWT-токены из прикладных запросов.

Маршрутизация запросов

Основной ресурс для управления маршрутизацией — `VirtualService`, он позволяет переопределять судьбу HTTP-или TCP-запроса. Доступные аргументы для принятия решения о маршрутизации:

- Host и любые другие заголовки;
- URI;
- метод (GET, POST и пр.);
- лейблы пода или неймспейса источника запросов;
- dst-IP или dst-порт для не-HTTP-запросов.

Управление балансировкой запросов между эндпоинтами сервиса

Основной ресурс для управления балансировкой запросов — `DestinationRule`, он позволяет настроить нюансы исходящих из подов запросов:

- лимиты/таймауты для TCP;
- алгоритмы балансировки между эндпоинтами;
- правила определения проблем на стороне эндпоинта для вывода его из балансировки;
- нюансы шифрования.



Все настраиваемые лимиты работают для каждого пода клиента по отдельности. Если настроить для сервиса ограничение на одно TCP-соединение, а клиентских подов — три, то сервис получит три входящих соединения.

Observability

Трассировка

Istio позволяет осуществлять сбор трейсов с приложений и инъекцию трассировочных заголовков, если таковых нет. При этом важно понимать следующее:

- Если запрос инициирует на сервисе вторичные запросы, для них необходимо наследовать трассировочные заголовки средствами приложения.
- Jaeger для сбора и отображения трейсов потребуется устанавливать самостоятельно.

Grafana

В стандартной комплектации с модулем предоставлены дополнительные дашборды:

- дашборд для оценки производительности и успешности запросов/ответов между приложениями;
- дашборд для оценки работоспособности и нагрузки на control plane.

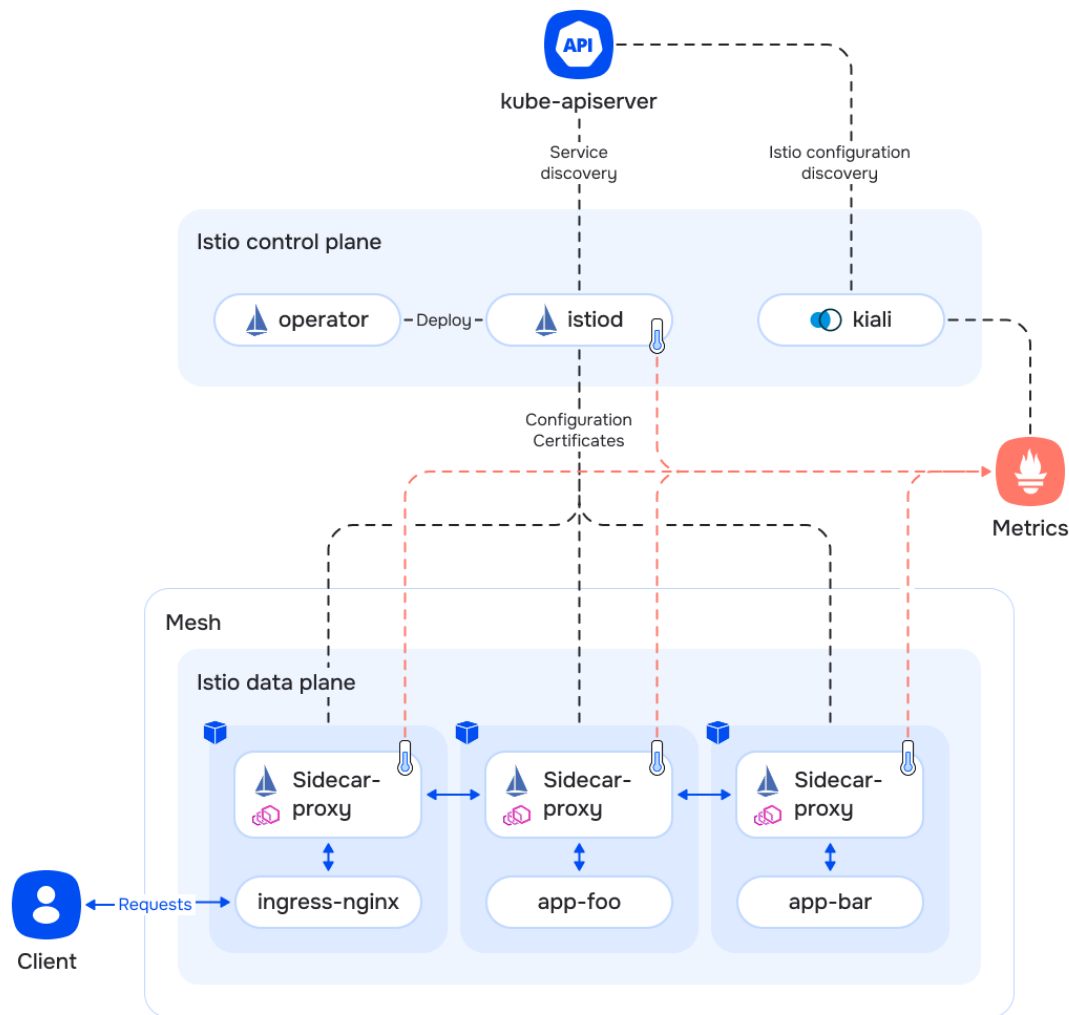
Kiali

Инструмент для визуализации дерева сервисов вашего приложения. Позволяет быстро оценить обстановку в сетевой связности благодаря визуализации запросов и их количественных характеристик непосредственно на схеме.

Архитектура кластера с включенным Istio

Компоненты кластера делятся на две категории:

- control plane — управляющие и обслуживающие сервисы. Под control plane обычно подразумевают поды istiod.
- data plane — прикладная часть Istio. Представляет собой контейнеры сайдкар-прокси.



Все сервисы из data plane группируются в mesh. Его характеристики:

- Общий неймспейс для генерации идентификатора сервиса в формате `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. Каждый mesh имеет идентификатор `TrustDomain`, который в нашем случае совпадает с доменом кластера. Например: `mycluster.local/ns/myns/sa/myapp`.
- Сервисы в рамках одного mesh имеют возможность аутентифицировать друг друга с помощью доверенных корневых сертификатов.

Элементы control plane:

- **istiod** — ключевой сервис, обеспечивающий решение следующих задач:
 - Непрерывная связь с API Kubernetes и сбор информации о прикладных сервисах.
 - Обработка и валидация с помощью механизма Kubernetes Validating Webhook всех Custom Resources, которые связаны с Istio.
 - Компоновка конфигурации для каждого сайдкар-прокси индивидуально:
 - генерация правил авторизации, маршрутизации, балансировки и прочих;
 - распространение информации о других прикладных сервисах в кластере;
 - выпуск индивидуальных клиентских сертификатов для организации схемы Mutual TLS. Эти сертификаты не связаны с сертификатами, которые использует и контролирует сам Kubernetes для своих служебных нужд.
 - Автоматическая подстройка манифестов, определяющих прикладные поды через механизм Kubernetes Mutating Webhook:
 - внедрение дополнительного служебного контейнера сайдкар-прокси;
 - внедрение дополнительного init-контейнера для адаптации сетевой подсистемы (настройка DNAT для перехвата прикладного трафика);
 - перенаправление readiness- и liveness-проб через сайдкар-прокси.
- **operator** — компонент, отвечающий за установку всех ресурсов, необходимых для работы control plane определенной версии.
- **kiali** — панель управления и наблюдения за ресурсами Istio и пользовательскими сервисами под управлением Istio, позволяющая следующее:
 - Визуализировать связи между сервисами.
 - Диагностировать проблемные связи между сервисами.
 - Диагностировать состояние control plane.

Для приема пользовательского трафика необходима доработка Ingress-контроллера:

- К подам контроллера добавляется сайдкар-прокси, который обслуживает только трафик от контроллера в сторону прикладных сервисов (параметр `IngressNginxController enableIstioSidecar` у ресурса `IngressNginxController`).

- Сервисы не под управлением Istio продолжают работать как раньше, запросы в их сторону не перехватываются сайдкар-контроллером.
- Запросы в сторону сервисов под управлением Istio перехватываются сайдкар-контроллером и обрабатываются в соответствии с правилами Istio (подробнее о том, как активировать Istio для приложения).

Контроллер istiod и каждый контейнер сайдкар-прокси экспортируют собственные метрики, которые собирает кластерный Prometheus.

Архитектура прикладного сервиса с включенным Istio

Особенности

- Каждый под сервиса получает дополнительный контейнер — сайдкар-прокси. Технически этот контейнер содержит два приложения:
 - **Envoy** — проксирует прикладной трафик и реализует все функции, которые предоставляет Istio, включая маршрутизацию, аутентификацию, авторизацию и пр.
 - **pilot-agent** — часть Istio, отвечает за поддержание конфигурации Envoy в актуальном состоянии, а также содержит в себе кеширующий DNS-сервер.
- В каждом поде настраивается DNAT входящих и исходящих прикладных запросов в сайдкар-прокси. Делается это с помощью дополнительного init-контейнера. Таким образом, трафик будет перехватываться прозрачно для приложений.
- Так как входящий прикладной трафик перенаправляется в сайдкар-прокси, readiness/liveness-трафика это тоже касается. Подсистема Kubernetes, которая за это отвечает, не рассчитана на формирование проб в формате Mutual TLS. Для адаптации все существующие пробы автоматически перенастраиваются на специальный порт в сайдкар-прокси, который перенаправляет трафик на приложение в неизменном виде.
- Для приема запросов извне кластера необходимо использовать подготовленный Ingress-контроллер:
 - Поды контроллера аналогично имеют дополнительный контейнер сайдкар-прокси.
 - В отличие от подов приложения, сайдкар-прокси Ingress-контроллера перехватывает только трафик от контроллера к сервисам. Входящий трафик от пользователей обрабатывает непосредственно сам контроллер.
- Ресурсы типа Ingress требуют минимальной доработки в виде добавления аннотаций:
 - `nginx.ingress.kubernetes.io/service-upstream: "true"` — Ingress-контроллер в качестве upstream будет использовать ClusterIP сервиса вместо адресов подов. Балансировкой трафика между подами теперь занимается сайдкар-прокси. Используйте эту опцию, только если у вашего сервиса есть ClusterIP.
 - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — сайдкар-прокси Ingress-контроллера принимает решения о маршрутизации на основе заголовка Host. Без данной аннотации контроллер оставит заголовок с адресом сайта, например `Host: example.com`.
- Ресурсы типа Service не требуют адаптации и продолжают выполнять свою функцию. Приложениям все так же доступны адреса сервисов вида `servicename`, `servicename.myns.svc` и пр.
- DNS-запросы изнутри подов прозрачно перенаправляются на обработку в сайдкар-прокси:
 - Требуется для разыменования DNS-имен сервисов из соседних кластеров.

Жизненный цикл пользовательского запроса

Приложение с выключенным Istio

Приложение с включенным Istio

Как активировать Istio для приложения

Основная цель активации — добавить сайдкар-контейнер к подам приложения, после чего Istio сможет управлять трафиком.

Рекомендованный способ добавления сайдкар-контейнера — использовать `sidecar-injector`. Istio умеет «подсаживать» к вашим подам сайдкар-контейнер с помощью механизма Admission Webhook. Настраивается с помощью лейблов и аннотаций:

- Лейбл к неймспейсу — обозначает ваш неймспейс для компонента `sidecar-injector`. После применения лейбла к новым подам будут добавлены сайдкар-контейнеры:
 - `istio-injection=enabled` — использует глобальную версию Istio (`spec.settings.globalVersion` в `ModuleConfig`);
 - `istio.io/rev=v1x21` — использует конкретную версию Istio для этого неймспейса;
 - `istio.io/rev=default` — использует глобальную версию Istio (`spec.settings.globalVersion` в `ModuleConfig`).
- Аннотация к поду `sidecar.istio.io/inject ("true" или "false")` позволяет локально переопределить политику `sidecarInjectorPolicy`. Эти аннотации работают только в неймспейсах, обозначенных лейблами из списка выше.

Также существует возможность добавить сайдкар к индивидуальному поду в неймспейсе без установленных лейблов `istio-injection=enabled` или `istio.io/rev=vXxYZ` путем установки лейбла `sidecar.istio.io/inject=true`.



Istio-прокси, который работает в качестве сайдкар-контейнера, тоже потребляет ресурсы и добавляет накладные расходы.

Накладные расходы, добавляемые Istio-прокси:

- Любой входящий запрос принудительно перехватывается прокси-сервером Envoy. Он анализирует его и отправляет дальше, используя DNAT-правила, создавая новое соединение. Со стороны получателя происходит то же самое: трафик сначала попадает в «принимающий» Envoy, а только потом — в само

приложение.

- Каждый Envoy хранит информацию обо всех сервисах в кластере, поэтому масштабирование кластера ведет к линейному росту потребления RAM в Envoy из-за хранения полной карты сервисов. С помощью кастомного ресурса Sidecar происходит фильтрация конфигурации и передача в Envoy только необходимого минимума данных.



Интерфейс EnvoyFilter может управляться плагинами на Lua, но является внутренним механизмом управления для реализации функциональности Istio. Использовать его в настройках пользовательских конфигураций запрещено. Это нарушит целостность системы.

Также важно подготовить Ingress-контроллер и Ingress-ресурсы приложения:

- Включите `enableIstioSidecar` у ресурса `IngressNginxController`.
- Добавьте аннотации на Ingress-ресурсы приложения:
 - `nginx.ingress.kubernetes.io/service-upstream: "true"` — Ingress-контроллер в качестве upstream использует ClusterIP сервиса вместо адресов подов. Балансировкой трафика между подами теперь занимается сайдкар-прокси. Используйте эту опцию, только если у вашего сервиса есть ClusterIP;
 - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — сайдкар-прокси Ingress-контроллера принимает решения о маршрутизации на основе заголовка Host. Без этой аннотации контроллер оставит заголовок с адресом сайта, например `Host: example.com`.

Федерация и мультикластер

Поддерживаются две схемы межкластерного взаимодействия:

- федерация;
- мультикластер.

Принципиальные отличия:

- Федерация объединяет суверенные кластеры:
 - у каждого кластера собственный неймспейс (для неймспейсов, сервисов и пр.);
 - доступ к отдельным сервисам между кластерами явно обозначен.
- Мультикластер объединяет созависимые кластеры:
 - неймспейс у кластеров общий — каждый сервис доступен для соседних кластеров так, словно он работает на локальном кластере (если это не запрещают правила авторизации).

Федерация

Требования к кластерам

- У каждого кластера должен быть уникальный домен в параметре `clusterDomain` ресурса `ClusterConfiguration`. Обратите внимание, что ни один из кластеров не должен иметь домен `cluster.local`, который является значением по умолчанию.



`cluster.local` — неизменяемый псевдоним для домена локального кластера. Указание `cluster.local` как `principals` в `AuthorizationPolicy` всегда будет указывать на локальный кластер, даже если в mesh существует кластер, у которого `clusterDomain` явно определен как `cluster.local` (источник — документация Istio).

- Требования к уникальности подсетей сервисов и подов в параметрах `serviceSubnetCIDR` и `podSubnetCIDR` ресурса `ClusterConfiguration` при работе кластеров в федерации нет.
 - При анализе HTTP- и HTTPS-запросов (в терминологии Istio) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно по заголовкам.
 - При анализе TCP-запросов (в терминологии Istio) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно только по IP-адресу назначения и номеру порта.



Если IP-адреса сервисов или подов пересекутся между кластерами, то под маршрутизирующие, запрещающие или разрешающие правила istio могут попасть запросы других подов иных кластеров. Пересечение подсетей сервисов и подов жестко запрещено в `single-network` режиме, и допустимо, но не рекомендуется в режиме `multi-networks` (источник — документация Istio).

Istio работает в режиме `multi-network` — поды разных кластеров взаимодействуют друг с другом только через Istio ingress gateway. Прямое взаимодействие между подами разных кластеров не поддерживается.

Общие принципы федерации

- Федерация требует установления взаимного доверия между кластерами. Соответственно, для установления федерации нужно в кластере А сделать кластер Б доверенным и, аналогично, в кластере Б сделать кластер А доверенным. Это достигается взаимным обменом корневыми сертификатами.
- Для прикладной эксплуатации федерации необходимо также обменяться информацией о публичных сервисах. Чтобы опубликовать сервис `bar` из кластера Б в кластере А, необходимо в кластере А создать ресурс `ServiceEntry`, который описывает публичный адрес ingress-gateway кластера Б.

Включение федерации

При включении федерации (параметр модуля `istio.federation.enabled = true`) происходит следующее:

- В кластер добавляется сервис `ingressgateway`, чья задача — проксировать mTLS-трафик извне кластера на прикладные сервисы.
- В кластер добавляется сервис, который экспортирует метаданные кластера наружу:
 - корневой сертификат Istio (доступен без аутентификации);
 - список публичных сервисов в кластере (доступен только для аутентифицированных запросов из соседних кластеров);
 - список публичных адресов сервиса `ingressgateway` (доступен только для аутентифицированных запросов из соседних кластеров).

Управление федерацией

Для построения федерации необходимо сделать следующее:

- В каждом кластере создать набор ресурсов `IstioFederation`, которые описывают все остальные кластеры.
 - После успешного автосогласования между кластерами, в ресурсе `IstioFederation` заполнятся разделы `status.metadataCache.public` и `status.metadataCache.private` служебными данными, необходимыми для работы федерации.
- Каждый ресурс (сервис), который считается публичным в рамках федерации, пометить лейблом `federation.istio.deckhouse.io/public-service: ""`.
 - В кластерах из состава федерации, для каждого сервиса создадутся соответствующие `ServiceEntry`, ведущие на `ingressgateway` оригинального кластера.



В разделе `.spec.ports` этих сервисов у каждого порта должно быть заполнено поле `name`.

Мультикластер

Требования к кластерам

- Домены кластеров в параметре `clusterDomain` ресурса `ClusterConfiguration` должны быть одинаковыми для всех членов мультикластера. По умолчанию значение параметра — `cluster.local`.
- Подсети сервисов и подов в параметрах `serviceSubnetCIDR` и `podSubnetCIDR` ресурса `ClusterConfiguration` должны быть уникальными для каждого кластера.
 - При анализе HTTP- и HTTPS-запросов (в терминологии Istio) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно по заголовкам.
 - При анализе TCP-запросов (в терминологии Istio) идентифицировать их и принять решение о дальнейшей маршрутизации, запрещении или разрешении возможно только по IP-адресу назначения и номеру порта.



Если IP-адреса сервисов или подов пересекутся между кластерами, то под маршрутизирующие, запрещающие или разрешающие правила Istio могут попасть запросы других подов иных кластеров. Пересечение подсетей сервисов и подов не рекомендуется (источник — документация Istio).

Istio работает в режиме `multi-network` — поды разных кластеров взаимодействуют друг с другом только через Istio ingress gateway. Прямое взаимодействие между подами разных кластеров не поддерживается.

Общие принципы

- Мультикластер требует установления взаимного доверия между кластерами. Соответственно, для построения мультикластера нужно в кластере А сделать кластер Б доверенным и в кластере Б сделать кластер А доверенным. Технически это достигается взаимным обменом корневыми сертификатами.
- Для сбора информации о соседних сервисах Istio подключается напрямую к API-серверу соседнего кластера. Данный модуль Deckhouse берет на себя организацию соответствующего канала связи.

Включение мультикластера

При включении мультикластера (параметр модуля `istio.multicluster.enabled = true`) происходит следующее:

- В кластер добавляется прокси для публикации доступа к API-серверу посредством стандартного Ingress-ресурса:
 - Доступ через данный публичный адрес ограничен авторизацией на основе Bearer-токенов, подписанных доверенными ключами. Обмен доверенными публичными ключами происходит автоматически средствами Deckhouse при взаимной настройке мультикластера.
 - Сам прокси имеет `read-only`-доступ к ограниченному набору ресурсов.
- В кластер добавляется сервис, который экспортирует метаданные кластера наружу:
 - Корневой сертификат Istio (доступен без аутентификации).
 - Публичный адрес, через который доступен API-сервер (доступен только для аутентифицированных запросов из соседних кластеров).
 - Список публичных адресов сервиса `ingressgateway` (доступен только для аутентифицированных запросов из соседних кластеров).

- Публичные ключи сервера для аутентификации запросов к API-серверу и закрытым метаданным (см. выше).

Управление мультикластером

Для сборки мультикластера необходимо в каждом кластере создать набор ресурсов `IstioMultiCluster`, которые описывают все остальные кластеры.

В случае проблем при работе с мультикластером необходимо проверить в каждом кластере:

1. Состояние ресурсов `IstioMultiCluster`. Для этого выполните команду `d8 k describe istiomulticlust cluster-name`. Важно, чтобы в статусе ресурса был указан `Root CA` и в поле `Public Last Fetch Timestamp` была свежий лейбл времени.
2. В поле `Ingress Gateways` ресурса `IstioMultiCluster` должен быть указан корректный адрес (IP или FQDN) `IngressGateway` второго кластера.
3. С помощью утилиты `istioctl` (как установить...):

```
istioctl remote-clusters -i d8-istio
NAME          SECRET                                STATUS    ISTIOD
cluster-b     d8-istio/istio-remote-secret-cluster-b syncd     istiod-v1x21-5c57d85b54-k8p17
```

Накладные расходы

Внедрение Istio повлечёт за собой дополнительные расходы ресурсов, как для **control-plane** (контроллер istiod), так и для **data-plane** (istio-сайдкары приложений).

control-plane

Контроллер istiod непрерывно наблюдает за конфигурацией кластера, компонует настройки для istio-сайдкаров data-plane и рассылает их по сети. Соответственно, чем больше приложений и их экземпляров, чем больше сервисов и чем чаще эта конфигурация меняется, тем больше требуется вычислительных ресурсов и больше нагрузка на сеть. При этом, поддерживается два подхода для снижения нагрузки на экземпляры контроллеров:

- горизонтальное масштабирование (настройка модуля `controlPlane.replicasManagement`) — чем больше экземпляров контроллеров, тем меньше экземпляров istio-сайдкаров обслуживать каждому из них и тем меньше нагрузка на CPU и на сеть.
- сегментация data-plane с помощью ресурса *Sidecar* (рекомендуемый подход) — чем меньше область видимости у отдельного istio-сайдкара, тем меньше требуется обновлять данных в data-plane и тем меньше нагрузка на CPU и на сеть.

Примерная оценка накладных расходов для экземпляра control-plane, который обслуживает 1000 сервисов и 2000 istio-сайдкаров — 1 vCPU и 1.5 GB RAM.

data-plane

На потребление ресурсов data-plane (istio-сайдкары) влияет множество факторов:

- количество соединений,
- интенсивность запросов,
- размер запросов и ответов,
- протокол (HTTP/TCP),
- количество ядер CPU,
- сложность конфигурации Service Mesh.

Примерная оценка накладных расходов для экземпляра istio-сайдкара — 0.5 vCPU на 1000 запросов/сек и 50 MB RAM.

istio-сайдкары также вносят задержку в сетевые запросы — примерно 2.5мс на запрос.

Настройки

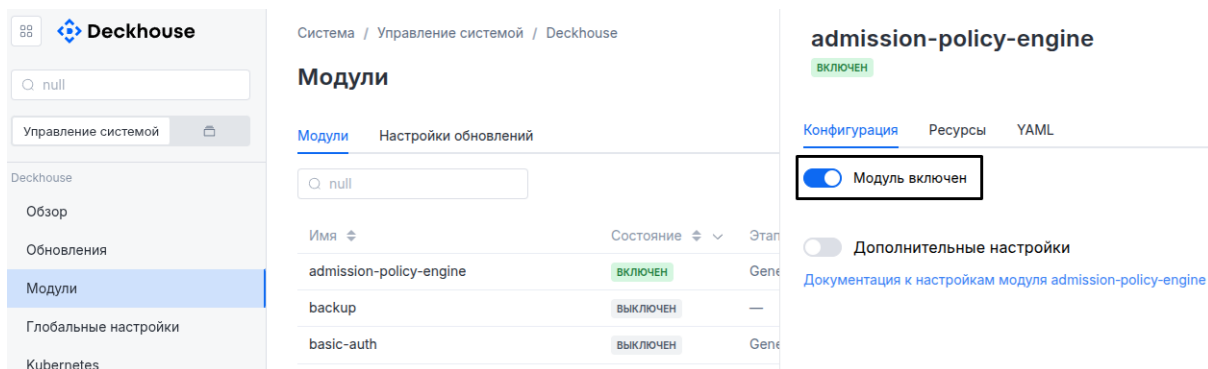
В модуле реализовано 17 алертов.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `istio`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable istio` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `istio`:

```
d8 system module enable istio
```

- **С помощью ModuleConfig `istio`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `istio` (создайте его, при необходимости).

Пример манифеста для включения модуля `istio`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  enabled: true
```

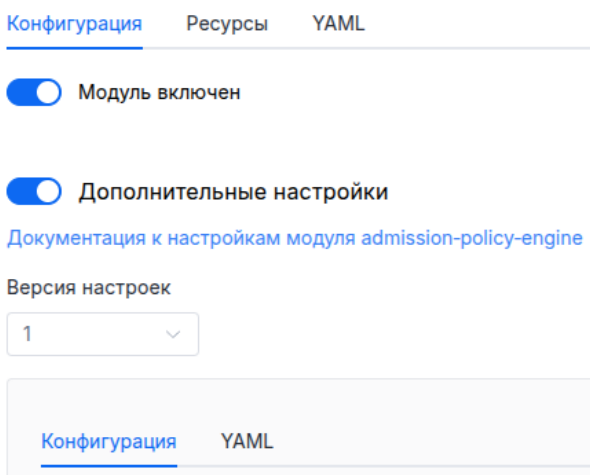
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `istio`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `istio` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `istio`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `istio` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc istio
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в

параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `istio`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
dk k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `istio`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  version: 3
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Удалите `.settings.auth.password`.

- Обновление из версии **2** в **3**:

- Перенесите `.settings.enableHTTP10` в `.settings.dataPlane.enableHTTP10`.
- Перенесите `.settings.proxyConfig` в `.settings.dataPlane.proxyConfig`.

Параметры

Версия схемы: 3

- **settings**

объект

- **settings.additionalVersions**

массив строк

Дополнительные версии control plane Istio, которые будут установлены. Они будут обслуживать паттерсы, где явно указана версия с помощью лейбла `istio.io/rev=`.

По умолчанию: `[]`

- **Элемент массива**

строка

Шаблон: `^[0-9]+\.[0-9]+$`

- **settings.alliance**

объект

Доступно в редакциях: EE, CSE Pro (1.73)

Общие настройки для федерации и мультикластера.

- **settings.alliance.ingressGateway**

объект

Доступно в редакциях: EE, CSE Pro (1.73)

Параметры для ingressgateway.

- **settings.alliance.ingressGateway.advertise**

массив объектов

Доступно в редакциях: EE, CSE Pro (1.73)

Фактические адреса, которые будут объявлены удаленным кластерам для организации межкластерных запросов приложений. Если не указано, адреса будут выявлены автоматически.

По умолчанию: `[]`

Примеры:


```
advertise:
- address: 172.16.0.5
  port: 15443
```

```
advertise:
- address: somehost.example.com
  port: 15443
```

■ **settings.alliance.ingressGateway.advertise.address**
строка

Обязательный параметр

Шаблон: `^([a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z]{2,}$`

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

Примеры:

```
address: somehost.example.com
```

```
address: 172.16.0.5
```

■ **settings.alliance.ingressGateway.advertise.port**
целочисленный

Обязательный параметр

Допустимые значения: `1024 <= X <= 65535`

■ **settings.alliance.ingressGateway.gatewayPodAnnotations**
объект

Доступно в редакциях: EE, CSE Pro (1.73)

Дополнительные аннотации для подов DaemonSet'a ingressgateway.

Пример:

```
gatewayPodAnnotations:
  checksum/config: xyz-123
```

■ **settings.alliance.ingressGateway.inlet**
строка

Доступно в редакциях: EE, CSE Pro (1.73)

Способ публикации ingressgateway:

- `LoadBalancer` — рекомендуется в случае, если площадка облачная и поддерживает LB;
- `NodePort` — для площадок без LB.

По умолчанию: `LoadBalancer`

Допустимые значения: `LoadBalancer`, `NodePort`

Пример:

```
inlet: LoadBalancer
```

■ **settings.alliance.ingressGateway.nodePort**
объект

Доступно в редакциях: EE, CSE Pro (1.73)

Специальные настройки для ingressgateway с инлетом `NodePort`.

Примеры:

```
nodePort: {}
```

```
nodePort:
  port: 30001
```

■ **settings.alliance.ingressGateway.nodePort.port**
целочисленный

Статичный порт для сервиса с типом NodePort. Должен быть из диапазона, заданного аргументом `kube-apiserver --service-node-port-range` (по умолчанию 30000-32767).

Допустимые значения: `1024 <= X <= 65535`

- **settings.alliance.ingressGateway.nodeSelector**
объект

Доступно в редакциях: EE, CSE Pro (1.73)

Селектор для DaemonSet'a ingressgateway.

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Пример:

```
nodeSelector:
  type: ingress
```

- **settings.alliance.ingressGateway.serviceAnnotations**
объект

Доступно в редакциях: EE, CSE Pro (1.73)

Дополнительные аннотации для сервиса ingressgateway.

Полезно, например, для настройки локального LB в Yandex Cloud (аннотация `yandex.cpi.flant.com/listener-subnet-id`).

Пример:

```
serviceAnnotations:
  yandex.cpi.flant.com/listener-subnet-id: xyz-123
```

- **settings.alliance.ingressGateway.tolerations**
массив объектов

Доступно в редакциях: EE, CSE Pro (1.73)

tolerations для DaemonSet'a ingressgateway.

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Пример:

```
tolerations:
- operator: Exists
```

- **settings.alliance.ingressGateway.tolerations.effect**
строка
- **settings.alliance.ingressGateway.tolerations.key**
строка
- **settings.alliance.ingressGateway.tolerations.operator**
строка
- **settings.alliance.ingressGateway.tolerations.tolerationSeconds**
целочисленный
- **settings.alliance.ingressGateway.tolerations.value**
строка

- **settings.auth**
объект

Опции, связанные с аутентификацией или авторизацией в приложении.

Пример:

```
auth:
  externalAuthentication:
    authURL: https://dex.d8.svc.cluster.local/dex/auth
    authSignInURL: https://example.com/dex/sign_in
  allowedUserGroups:
  - admins
```

- **settings.auth.allowedUserEmails**
массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.allowedUserGroups**
массив строк

Массив групп, пользователям которых разрешён доступ в публичные веб-интерфейсы модуля.

Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.

Внимание! При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках `DexProvider`.

- **settings.auth.externalAuthentication**

объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX external-auth, работающий на основе модуля Nginx auth_request).

i

Внешняя аутентификация включается автоматически, если включен модуль user-authn.

- **settings.auth.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

Пример:

```
authSignInURL: https://example.com/dex/sign_in
```

- **settings.auth.externalAuthentication.authURL**
строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

Пример:

```
authURL: https://example.com/dex/auth
```

- **settings.auth.satisfyAny**
булевый

Разрешает пройти только одну из аутентификаций.

В комбинации с опцией `whitelistSourceRanges` позволяет считать авторизованными всех пользователей из указанных сетей без ввода логина и пароля.

По умолчанию: `false`

Пример:

```
satisfyAny: true
```

- **settings.auth.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить авторизацию в публичные веб-интерфейсы модуля.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **settings.ca**
объект

Явно заданный корневой сертификат, который будет использован для подписывания индивидуальных сертификатов сервисов, в случае включения mTLS.

Для создания сертификата можно воспользоваться примером, в котором `basicConstraints = CA:FALSE` необходимо заменить на `basicConstraints = CA:TRUE`.

- **settings.ca.cert**
строка

Корневой или промежуточный сертификат в формате PEM.

- **settings.ca.chain**
строка

Цепочка сертификатов в формате PEM на случай, если `cert` — промежуточный сертификат.

- **settings.ca.key**
строка

Ключ корневого сертификата в формате PEM.

- **settings.ca.root**
строка

Корневой сертификат в формате PEM на случай, если `cert` — промежуточный сертификат.

- **settings.controlPlane**
объект

Настройки для компонента istiod.

- **settings.controlPlane.nodeSelector**

объект

Опциональный селектор для компонента istiod. Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.controlPlane.replicasManagement**

объект

Настройки управления репликами и горизонтальным масштабированием istiod.

Примеры:

```
replicasManagement:
  mode: Standard
```

```
replicasManagement:
  mode: Static
  static:
    replicas: 3
```

```
replicasManagement:
  mode: HPA
  hpa:
    minReplicas: 2
    maxReplicas: 5
    metrics:
      - type: CPU
        targetAverageUtilization: 80
```

- **settings.controlPlane.replicasManagement.hpa**

объект

Параметры управления репликами и масштабированием в режиме HPA.

- **settings.controlPlane.replicasManagement.hpa.maxReplicas**

число

Обязательный параметр

Максимальное количество реплик, которое может быть установлено HPA. Не может быть меньше `minReplicas`.

Допустимые значения: `1 <= X`

- **settings.controlPlane.replicasManagement.hpa.metrics**

массив объектов

Обязательный параметр

HPA будет основываться на этих метриках при принятии решения об увеличении или уменьшении количества реплик.

- **settings.controlPlane.replicasManagement.hpa.metrics.targetAverageUtilization**

число

Обязательный параметр

Целевое значение средней загрузки CPU во всех репликах. Задается в процентах от `Requests CPU`.

Допустимые значения: `1 <= X <= 100`

- **settings.controlPlane.replicasManagement.hpa.metrics.type**

строка

Обязательный параметр

Тип метрики.

Допустимые значения: `CPU`

- **settings.controlPlane.replicasManagement.hpa.minReplicas**

число

Обязательный параметр

Минимальное количество реплик, которое может быть установлено HPA.

Допустимые значения: `1 <= X`

- **settings.controlPlane.replicasManagement.mode**

строка

Режим работы с репликами:

- **Standard** — режим управления репликами и масштабированием в соответствии с глобальным режимом отказоустойчивости (параметр `highAvailability`);
- **Static** — режим, где количество реплик указывается явно (параметр `static.replicas`);
- **HPA** — режим, где количество реплик рассчитывается автоматически с помощью HPA на основе загрузки CPU. Настраивается в секции параметров `hpa`.

По умолчанию: `Standard`

Допустимые значения: `Standard`, `Static`, `HPA`

■ **settings.controlPlane.replicasManagement.static**
объект

Параметры управления репликами и масштабированием в режиме `Static`.

■ **settings.controlPlane.replicasManagement.static.replicas**
число

Обязательный параметр

Желаемое количество реплик.

Допустимые значения: `1 <= X`

■ **settings.controlPlane.resourcesManagement**
объект

Настройки запросов (`requests`) и ограничений (`limits`) использования CPU и памяти подами `istiod`.

Примеры:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: '2'
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

■ **settings.controlPlane.resourcesManagement.mode**
строка

Режим управления ресурсами:

- **Static** — с помощью прямого указания запросов/ограничений (`requests/limits`). Настраивается в секции параметров `static`;
- **VPA** — с помощью VPA. Настраивается в секции параметров `vpa`.

По умолчанию: `VPA`

Допустимые значения: `VPA`, `Static`

- **settings.controlPlane.resourcesManagement.static**
объект

Настройка управления ресурсами в режиме `Static`.

- **settings.controlPlane.resourcesManagement.static.limits**
объект

Настройка ограничений (limits) использования CPU и памяти.

- **settings.controlPlane.resourcesManagement.static.limits.cpu**
строка

Настройка ограничений использования CPU (CPU limits).

Шаблон: `^[0-9]+m?$`

- **settings.controlPlane.resourcesManagement.static.limits.memory**
строка от число

Настройка ограничений использования памяти (memory limits).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.static.requests**
объект

Настройки запросов ресурсов (requests) подов.

- **settings.controlPlane.resourcesManagement.static.requests.cpu**
строка

Настройка запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

- **settings.controlPlane.resourcesManagement.static.requests.memory**
строка от число

Настройка запроса памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.vpa**
объект

Параметры управления ресурсами в режиме VPA.

- **settings.controlPlane.resourcesManagement.vpa.cpu**
объект

Настройки VPA при работе с CPU.

- **settings.controlPlane.resourcesManagement.vpa.cpu.limitRatio**
число

Коэффициент расчета ограничений использования CPU относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования CPU для пода (CPU limits).

Если параметр указан, то VPA, при пересчете запросов CPU (CPU requests) будет пересчитывать ограничения использования CPU (CPU limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.controlPlane.resourcesManagement.vpa.cpu.max**
строка от число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `2`

Шаблон: `^[0-9]+m?$`

- **settings.controlPlane.resourcesManagement.vpa.cpu.min**
строка от число

Минимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `50m`

Шаблон: `^[0-9]+m?$`

- **settings.controlPlane.resourcesManagement.vpa.memory**
объект

Настройки VPA при работе с памятью.

строка

Тестовый формат access-логов sidecar-контейнера. Операторы шаблона описаны в документации envoy.

По умолчанию: `[%START TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%" %RESPONSE CODE% %RESPONSE FLAGS% %RESPONSE CODE DETAILS% %CONNECTION TERMINATION DETAILS% "%UPSTREAM TRANSPORT FAILURE REASON%" %BYTES RECEIVED% %BYTES SENT% %DURATION% %RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%" "%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM HOST%" %UPSTREAM LOCAL ADDRESS% %DOWNSTREAM LOCAL_ADDRESS% %DOWNSTREAM_REMOTE_ADDRESS% %REQUESTED_SERVER_NAME% %ROUTE_NAME%`

■ **settings.dataPlane.accessLog.type**

строка

По умолчанию: `Text`

Допустимые значения: `Text`, `JSON`

■ **settings.dataPlane.enableHTTP10**

булевый

Обрабатывать ли HTTP/1.0-запросы в sidecar-контейнерах или отвергать их с кодом ответа `426 Upgrade Required`.

По умолчанию: `false`

Пример:

```
enableHTTP10: true
```

■ **settings.dataPlane.extensionProviders**

массив объектов

Дополнительные Istio extension providers, которые будут зарегистрированы в mesh configuration.

Имя provider'a `main-access-log-format` зарезервировано Deckhouse.

По умолчанию: `[]`

Пример:

```
extensionProviders:
- name: authservice-grpc
  envoyExtAuthzGrpc:
    service: authservice.example.svc.cluster.local
    port: 10003
```

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc**

объект

Обязательный параметр

Внешний сервис авторизации, который реализует Envoy ext_authz gRPC API.

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.clearRouteCache**

булевый

Очищать route cache, чтобы внешний сервис авторизации мог влиять на маршрутизацию.

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.failOpen**

булевый

Разрешать запросы, если сервис авторизации недоступен или вернул ошибку 5xx.

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.port**

целочисленный

Обязательный параметр

Порт сервиса.

Допустимые значения: `1 <= X <= 65535`

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.service**

строка

Обязательный параметр

Сервис, который реализует Envoy ext_authz gRPC API. Формат значения — `[<hostname>]/[<namespace>]`.

■ **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.statusOnError**

строка

HTTP-статус, который будет возвращён клиенту при ошибке взаимодействия с сервисом

авторизации.

- **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.timeout**
строка

Максимальное время ожидания ответа от сервиса авторизации.

- **settings.dataPlane.extensionProviders.name**
строка

Обязательный параметр

Уникальное имя extension provider'a.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.dataPlane.proxyConfig**
объект

Централизованная настройка ProxyConfig.

- **settings.dataPlane.proxyConfig.holdApplicationUntilProxyStarts**
булевый

Этот параметр заставляет sidecar-injector вставлять sidecar-контейнер на первое место в списке контейнеров пода. Также он добавляет postStart-хук для того, чтобы убедиться, что Envoy инициализировался раньше самого приложения. Это позволит избежать сетевых ошибок при старте пода.

Этот глобальный параметр может быть переопределен на уровне пода аннотацией `proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts": true }'`.

По умолчанию: `false`

Пример:

```
holdApplicationUntilProxyStarts: true
```

- **settings.dataPlane.proxyConfig.idleTimeout**
строка

Тайм-аут для соединений без прикладной активности, которые установлены между istio-сайдкаром клиента и сервисом. По истечении тайм-аута, соединение между сайдкаром и сервисом закрывается, но между приложением и сайдкаром — не закрывается. Если установить значение `0s`, то тайм-аут будет отключен. Этот глобальный параметр может быть переопределен на уровне пода аннотацией:

```
proxy.istio.io/config: |-
  proxyMetadata:
    ISTIO_META_IDLE_TIMEOUT: "12h"
```



Внимание. Отключение этого таймаута (значение `0s`) с большой вероятностью может привести к утечке соединений из-за потери пакетов TCP FIN и т.п. **Внимание.** После изменения настройки необходим рестарт клиентских подов.

По умолчанию: `1h`

Шаблон: `^[0-9]+(s|m|h)$`

Пример:

```
idleTimeout: 24h
```

- **settings.dataPlane.trafficRedirectionSetupMode**
строка

Управление режимом перенаправления прикладного трафика для передачи под управление Istio в сетевом пространстве имён пода.

- **CNIPlugin** — в этом режиме настройку осуществляет CNI plugin при создании пода на узле. Этот режим не требует дополнительных прав для пода и является рекомендуемым. Режим имеет ограничения при использовании прикладных init-контейнеров, которые осуществляют сетевое взаимодействие с другими сервисами.
- **InitContainer** — классический режим, при котором каждый под автоматически снабжается специальным init-контейнером, который осуществляет настройку сетевого окружения пода. Для того, чтобы осуществить эту настройку, контейнеру выдаются дополнительные права, что может не соответствовать требованиям безопасности в отдельных инсталляциях.

По умолчанию: `InitContainer`

Допустимые значения: `CNIPlugin`, `InitContainer`

Примеры:

```
trafficRedirectionSetupMode: CNIPlugin
```

```
trafficRedirectionSetupMode: InitContainer
```

■ **settings.dataPlane.ztunnel**

объект

Настройки для компонента ztunnel.

■ **settings.dataPlane.ztunnel.resourcesManagement**

объект

Настройки запросов (requests) и ограничений (limits) использования CPU и памяти подами istiod.

Примеры:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: '2'
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

■ **settings.dataPlane.ztunnel.resourcesManagement.mode**

строка

Режим управления ресурсами:

- **Static** — с помощью прямого указания запросов/ограничений (requests/limits). Настраивается в секции параметров static;
- **VPA** — с помощью VPA. Настраивается в секции параметров vpa.

По умолчанию: **VPA**

Допустимые значения: **VPA**, **Static**

■ **settings.dataPlane.ztunnel.resourcesManagement.static**

объект

Настройка управления ресурсами в режиме **Static**.

■ **settings.dataPlane.ztunnel.resourcesManagement.static.limits**

объект

Настройка ограничений (limits) использования CPU и памяти.

- **settings.dataPlane.ztunnel.resourcesManagement.static.limits.cpu**
строка

Настройка ограничений использования CPU (CPU limits).

Шаблон: `^[0-9]+m?$`

- **settings.dataPlane.ztunnel.resourcesManagement.static.limits.memory**
строка от число

Настройка ограничений использования памяти (memory limits).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.dataPlane.ztunnel.resourcesManagement.static.requests**
объект

Настройки запросов ресурсов (requests) подов.

- **settings.dataPlane.ztunnel.resourcesManagement.static.requests.cpu**
строка

Настройка запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

- **settings.dataPlane.ztunnel.resourcesManagement.static.requests.memory**
строка от число

Настройка запроса памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa**
объект

Параметры управления ресурсами в режиме VPA.

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu**
объект

Настройки VPA при работе с CPU.

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.limitRatio**
число

Коэффициент расчета ограничений использования CPU относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования CPU для пода (CPU limits).

Если параметр указан, то VPA, при пересчете запросов CPU (CPU requests) будет пересчитывать ограничения использования CPU (CPU limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.max**
строка от число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: 2

Шаблон: `^[0-9]+m?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.min**
строка от число

Минимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: 50m

Шаблон: `^[0-9]+m?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory**
объект

Настройки VPA при работе с памятью.

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.limitRatio**
число

Коэффициент расчета ограничений использования памяти относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования памяти для пода (memory limits).

Если параметр указан, то VPA, при пересчете запросов памяти (memory requests) будет пересчитывать ограничения использования памяти (memory limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.max**
строка or число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: 2Gi

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.min**
строка or число

Минимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: 256Mi

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa.mode**
строка

Режим работы VPA.

- **Initial** — VPA устанавливает начальные значения requests/limits при создании пода. В дальнейшем значения ресурсов не изменяются автоматически.
- **InPlaceOrRecreate** — VPA пытается изменить ресурсы пода «на месте» (in-place), если это поддерживается кластером. Если in-place обновление невозможно, под будет пересоздан.
- **Auto** — VPA автоматически пересоздаёт поды для применения новых значений ресурсов. Начиная с версии Deckhouse 1.75 данный режим считается устаревшим. Рекомендуется использовать **InPlaceOrRecreate**.

По умолчанию: **InPlaceOrRecreate**

Допустимые значения: **Initial**, **InPlaceOrRecreate**, **Auto**

- **settings.federation**
объект

Доступно в редакциях: EE, CSE Pro (1.73)

Параметры федерации с другими кластерами.

- **settings.federation.enabled**
булевый

Обозначить данный кластер как участник федерации (см. Включение федерации).

По умолчанию: **false**

Пример:

```
enabled: true
```

- **settings.globalVersion**
строка

Явно заданная версия control plane Istio, который обслуживает data plane по умолчанию (namespace с лейблом `istio-injection=enabled`, но не `istio.io/rev=`).

По умолчанию: 1.25

Шаблон: `^[0-9]+\.[0-9]+$`

- **settings.highAvailability**
булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

- **settings.https**
объект

Тип сертификата, используемого для публичных веб-интерфейсов модуля.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.https.certManager**

объект

■ **settings.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

■ **settings.https.customCertificate**

объект

■ **settings.https.customCertificate.secretName**

строка

Имя Secret'a в namespace `d8-system`, который будет использоваться для Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

Secret должен быть в формате `kubernetes.io/tls`.

По умолчанию: `false`

■ **settings.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy будут работать по HTTPS и заказывать сертификат с помощью ClusterIssuer заданном в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy будут работать по HTTPS, используя сертификат из namespace `d8-system`.
- `OnlyInURI` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy будут работать по HTTP (подразумевая, что перед ними стоит внешний балансировщик, который терминирует HTTPS) и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

Важно! В отличие от остальных модулей, istio не поддерживает работу без использования HTTPS (`mode: Disabled`).

По умолчанию: `CertManager`

Допустимые значения: `CertManager`, `CustomCertificate`, `OnlyInURI`

○ **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для Kiali, metadata-exporter и api-proxy.

Оptionальный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

○ **settings.jwksResolverAdditionalRootCA**

строка

Дополнительный корневой сертификат в формате PEM. Используется компонентом istiod при получении JWKS URI по HTTPS.

Пример: `-----BEGIN CERTIFICATE----- ... -----END CERTIFICATE-----`

○ **settings.multicluster**

объект

Доступно в редакциях: EE, CSE Pro (1.73)

Параметры мультикластерной инсталляции.

■ **settings.multicluster.enabled**

булевый

Обозначить данный кластер как участник мультикластера (см. Включение мультикластера).

По умолчанию: `false`

Пример:

```
enabled: true
```

◦ **settings.nodeSelector**
объект

Опциональный селектор для компонентов istio-operator, metadata-exporter и Kiali.

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

◦ **settings.outboundTrafficPolicyMode**
строка

Как поступать с запросами в сторону внешних, то есть не учтенных в mesh'е сервисов:

- `AllowAny` — пропускать;
- `RegistryOnly` — запрещать. В данном случае для работы с внешними сервисами потребуется регистрировать их с помощью ресурса `ServiceEntry` или организовывать `egressgateway`.

По умолчанию: `AllowAny`

Допустимые значения: `AllowAny`, `RegistryOnly`

Пример:

```
outboundTrafficPolicyMode: AllowAny
```

◦ **settings.sidecar**
объект

Настройки для перехвата сетевого трафика в istio-sidecar.

▪ **settings.sidecar.excludeInboundPorts**
массив строк

Диапазоны входящих портов, чей трафик гарантированно не заворачивается через Istio.

Можно переопределить этот параметр для отдельного пода с помощью аннотации `traffic.sidecar.istio.io/excludeInboundPorts`.

По умолчанию: `[]`

Пример:

```
excludeInboundPorts:
- '8080'
- '8443'
```

▪ **Элемент массива**
строка

Шаблон: `^[0-9]{1,5}$`

▪ **settings.sidecar.excludeOutboundIPRanges**
массив строк

Трафик в сторону этих диапазонов IP гарантированно не заворачивается через Istio.

Можно переопределить этот параметр для отдельного пода с помощью аннотации `traffic.sidecar.istio.io/excludeOutboundIPRanges`.

По умолчанию: `[]`

Пример:

```
excludeOutboundIPRanges:
- 10.1.1.0/24
```

▪ **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

▪ **settings.sidecar.excludeOutboundPorts**
массив строк

Диапазоны исходящих портов, чей трафик гарантированно не заворачивается через Istio.

Можно переопределить этот параметр для отдельного пода с помощью аннотации `traffic.sidecar.istio.io/excludeOutboundPorts`.

По умолчанию: `[]`

Пример:

```
excludeOutboundPorts:
- '8080'
- '8443'
```

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,5}$`

- **settings.sidecar.includeOutboundIPRanges**
массив строк

Трафик в сторону этих диапазонов IP безоговорочно заворачивается через Istio.

Можно переопределить этот параметр для отдельного пода с помощью аннотации `traffic.sidecar.istio.io/includeOutboundIPRanges`.

По умолчанию: `["0.0.0.0/0"]`

Пример:

```
includeOutboundIPRanges:
- 10.1.1.0/24
```

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **settings.sidecar.resourcesManagement**
объект

Управляет ресурсами sidecar-контейнера Istio.

Внимание! Настройка применяется только для новых подов с `istio-proxy`.

Пример:

```
resourcesManagement:
  static:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 2000m
      memory: 1Gi
```

- **settings.sidecar.resourcesManagement.mode**
строка

Режим управления ресурсами:

- **Static** — с помощью прямого указания запросов/ограничений (requests/limits).
Настраивается в секции параметров static;

По умолчанию: `Static`

Допустимые значения: `Static`

- **settings.sidecar.resourcesManagement.static**
объект

Настройка управления ресурсами в режиме `Static`.

- **settings.sidecar.resourcesManagement.static.limits**
объект

Настройка ограничений (limits) использования CPU и памяти.

- **settings.sidecar.resourcesManagement.static.limits.cpu**
строка

Настройка ограничений использования CPU (CPU limits).

По умолчанию: `2000m`

Шаблон: `^[0-9]+m?$`

- **settings.sidecar.resourcesManagement.static.limits.memory**
строка or число

Настройка ограничений использования памяти (memory limits).

По умолчанию: `1Gi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.sidecar.resourcesManagement.static.requests**
объект

Настройки запросов ресурсов (requests) подов.

- **settings.sidecar.resourcesManagement.static.requests.cpu**
строка

Настройка запроса CPU (CPU requests).

По умолчанию: `100m`

Шаблон: `^[0-9]+m?$`

- **settings.sidecar.resourcesManagement.static.requests.memory**
строка or число

Настройка запроса памяти (memory requests).

По умолчанию: `128Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.tolerations**
массив объектов

Опциональные tolerations для компонентов istio-operator, metadata-exporter и Kiali.

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

- **settings.tracing**
объект

Параметры трассировки.

- **settings.tracing.collector**
объект

Параметры отправки трейсов в коллектор.

- **settings.tracing.collector.zipkin**
объект

Параметры протокола zipkin, который использует Istio для отправки трейсов.

Jaeger поддерживает этот протокол.

- **settings.tracing.collector.zipkin.address**
строка

Сетевой адрес в формате `<IP of FQDN>:<port>`, по которому доступен zipkin-коллектор.

Шаблон: `[0-9a-zA-Z\.-]+`

Пример:

```
address: zipkin.myjaeger.svc:9411
```

- **settings.tracing.enabled**
булевый

Включить отправку трейсов в коллектор и интегрировать их отображение в Kiali.

По умолчанию: `false`

Пример:

```
enabled: true
```

- **settings.tracing.kiali**
объект

Настройки отображения span'ов в Kiali.

Если настройки не предоставлены, Kiali не будет отображать трассировку.

Примеры:


```
kiali: {}
```

```
kiali:
  jaegerURLForUsers: https://tracing-service:4443/jaeger
  jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
```

■ **settings.tracing.kiali.jaegerGRPCEndpoint**
строка

Адрес GRPC-интерфейса Jaeger'a в формате `<proto>://<fqdn>[:port]/`, доступный сервису Kiali изнутри кластера для служебных запросов.

Если не указать, то Kiali не будет интерпретировать трейсы самостоятельно и ограничится генерацией ссылок на `jaegerURLForUsers`.

Пример:

```
jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
```

■ **settings.tracing.kiali.jaegerURLForUsers**
строка

Адрес UI Jaeger, доступный пользователю в формате `<proto>://<fqdn>[:port]/<base path>`.

Обязательный параметр.

Пример:

```
jaegerURLForUsers: https://tracing-service:4443/jaeger
```

■ **settings.tracing.sampling**
число

Параметр `sampling` используется для определения процента запросов, которые передаются в систему трассировки.

Этот параметр должен быть настроен в зависимости от трафика в сети и количества данных трассировки, которые необходимо собрать.

Параметр можно переопределить для каждого пода индивидуально с помощью следующей аннотации:

```
proxy.istio.io/config: |
  tracing:
    sampling: 100.0
```

По умолчанию: `1.0`

Допустимые значения: `0.01 <= X <= 100.0`

Пример:

```
sampling: 50.05
```

Аутентификация

По умолчанию используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication` (см. ниже). Если эти варианты отключены, модуль включит `basic auth` со сгенерированным паролем.

Для просмотра сгенерированного пароля используйте команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values istio -O json | jq '.istio.internal.auth.password'
```

Чтобы сгенерировать новый пароль, удалите секрет:

```
d8 k -n d8-istio delete secret/kiali-basic-auth
```



Параметр `auth.password` больше не поддерживается.

Кастомные ресурсы

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

IngressIstioController

Scope: Cluster

Version: v1alpha1

- **spec**
объект

Обязательный параметр

- **spec.hostPort**
объект

Секция настроек для инлета `HostPort`.

- **spec.hostPort.httpPort**
целочисленный

Порт для небезопасного подключения по HTTP.

Если параметр не указан, возможность подключения по HTTP отсутствует.

Обязательный параметр, если не указан `httpsPort`.

Пример:

```
httpPort: 80
```

- **spec.hostPort.httpsPort**
целочисленный

Порт для безопасного подключения по HTTPS.

Если параметр не указан, возможность подключения по HTTPS отсутствует.

Обязательный параметр, если не указан `httpPort`.

Пример:

```
httpsPort: 443
```

- **spec.ingressGatewayClass**
строка

Обязательный параметр

Имя Ingress-gateway класса для обслуживания Istio ingressgateway controller.

Значение данного параметра используется в качестве селектора лейблов ресурса Gateway.

Пример лейбла ресурса Gateway: `istio.deckhouse.io/ingress-gateway-class: <ingressGatewayClass value>`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Пример:

```
ingressGatewayClass: istio
```

- **spec.inlet**
строка

Обязательный параметр

Способ поступления трафика из внешнего мира:

- `LoadBalancer` — устанавливается Ingress-контроллер и заказывается Service с типом `LoadBalancer`;
- `NodePort` — устанавливается Ingress-контроллер, который доступен на портах узлов через `NodePort`;
- `HostPort` — устанавливается Ingress-контроллер, который доступен на портах узлов через `HostPort`.

Внимание! На одном хосте может быть только один контроллер с данным типом инлета.

Допустимые значения: `LoadBalancer`, `HostPort`, `NodePort`

- **spec.loadBalancer**
объект

Необязательный параметр.

Секция настроек для инлета `LoadBalancer`.

- **spec.loadBalancer.annotations**

- объект

- Аннотации, которые будут проставлены сервису для гибкой настройки балансировщика.

- **spec.nodePort**

- объект

- Секция настроек для инлета `NodePort`.

- **spec.nodePort.httpPort**

- целочисленный

- Порт для небезопасного подключения по HTTP.

- Если параметр не указан, возможность подключения по HTTP отсутствует.

- Обязательный параметр**, если не указан `httpsPort`.

- Пример:

```
httpPort: 30080
```

- **spec.nodePort.httpsPort**

- целочисленный

- Порт для безопасного подключения по HTTPS.

- Если параметр не указан, возможность подключения по HTTPS отсутствует.

- Обязательный параметр**, если не указан `httpPort`.

- Пример:

```
httpsPort: 30443
```

- **spec.nodeSelector**

- объект

- Как в `spec.nodeSelector` у подов.

- Если значение не указано или указано `false`, будет использоваться автоматика.

- Формат:** стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- **spec.resourcesRequests**

- объект

- Настройки максимальных значений CPU и памяти, которые может запросить под при выборе узла (если VPA выключен, максимальные значения становятся значениями по умолчанию).

- **spec.resourcesRequests.mode**

- строка

- Обязательный параметр

- Режим управления реквестами ресурсов.

- По умолчанию: `VPA`

- Допустимые значения: `VPA`, `Static`

- **spec.resourcesRequests.static**

- объект

- Настройки статического режима управления.

- **spec.resourcesRequests.static.cpu**

- строка

- Значение для реквеста к CPU.

- По умолчанию: `350m`

- **spec.resourcesRequests.static.memory**

- строка

- Значение для реквеста к памяти.

- По умолчанию: `500Mi`

- **spec.resourcesRequests.vpa**

- объект

- Настройки режима работы VPA.

- **spec.resourcesRequests.vpa.cpu**

- объект

- Настройки для CPU.

■ **spec.resourcesRequests.vpa.cpu.max**

строка

Максимальное значение, которое может выставить VPA для реквеста к CPU.

По умолчанию: `100m`■ **spec.resourcesRequests.vpa.cpu.min**

строка

Минимальное значение, которое может выставить VPA для реквеста к CPU.

По умолчанию: `50m`■ **spec.resourcesRequests.vpa.memory**

объект

Значение для реквеста к памяти.

■ **spec.resourcesRequests.vpa.memory.max**

строка

Максимальное значение, которое может выставить VPA для реквеста к памяти.

По умолчанию: `200Mi`■ **spec.resourcesRequests.vpa.memory.min**

строка

Минимальное значение, которое может выставить VPA для реквеста к памяти.

По умолчанию: `100Mi`■ **spec.resourcesRequests.vpa.mode**

строка

Режим работы VPA.

По умолчанию: `Initial`Допустимые значения: `Initial`, `Auto`○ **spec.tolerations**

массив объектов

Как в `spec.tolerations` у подов.Если значение не указано или указано `false`, будет использоваться автоматика.**Формат:** стандартный список toleration. Поды инстанса унаследуют это поле как есть.■ **spec.tolerations.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`■ **spec.tolerations.key**

строка

■ **spec.tolerations.operator**

строка

По умолчанию: `Equal`Допустимые значения: `Exists`, `Equal`■ **spec.tolerations.tolerationSeconds**

целочисленный

■ **spec.tolerations.value**

строка

IstioFederation

Scope: Cluster

Version: v1alpha1

Ресурс для настройки удалённого кластера как доверенного.

• **spec**

объект

Обязательный параметр

Доступно в редакциях: EE, CSE Pro (1.73)○ **spec.metadata**

объект

■ **spec.metadata.ca**

строка

Доступно в редакциях: EE, CSE Pro (1.73)

Сертификат для проверки HTTPS URL, который используется для публикации метаданных удаленного кластера.

Пример:

```
ca: "-----BEGIN CERTIFICATE----- ..... -----END CERTIFICATE-----"
```

- **spec.metadata.insecureSkipVerify**
булевый

Доступно в редакциях: EE, CSE Pro (1.73)

Исключение проверки HTTPS-сертификата на стороне клиента для URL, который используется для публикации метаданных удаленного кластера.

По умолчанию: `false`

Пример:

```
insecureSkipVerify: true
```

- **spec.metadataEndpoint**
строка

Обязательный параметр

Доступно в редакциях: EE, CSE Pro (1.73)

HTTPS-эндпоинт, который используется для публикации метаданных удаленного кластера.

Пример:

```
metadataEndpoint: https://istio.k8s.example.com/metadata/
```

- **spec.trustDomain**
строка

Обязательный параметр

Доступно в редакциях: EE, CSE Pro (1.73)

TrustDomain удаленного кластера.

Указание этого параметра является обязательным, однако на данный момент он не используется, так как текущая версия Istio не поддерживает привязку TrustDomain к корневому сертификату центра сертификации (CA).

Шаблон: `^[0-9a-zA-Z._-]+$`

Пример:

```
trustDomain: cluster.local
```

- **status**
объект

- **status.metadataCache**
объект

- **status.metadataCache.private**
объект

- **status.metadataCache.private.ingressGateways**

массив объектов

- **status.metadataCache.private.ingressGateways.address**

строка

- **status.metadataCache.private.ingressGateways.port**

целочисленный

- **status.metadataCache.private.publicServices**

массив объектов

- **status.metadataCache.private.publicServices.hostname**

строка

- **status.metadataCache.private.publicServices.ports**

массив объектов

- **status.metadataCache.private.publicServices.ports.name**

строка

- **status.metadataCache.private.publicServices.ports.port**

целочисленный

- **status.metadataCache.private.publicServices.ports.protocol**

строка

- **status.metadataCache.privateLastFetchTimestamp**

строка

- **status.metadataCache.public**

объект

- **status.metadataCache.public.authnKeyPub**

строка

- **status.metadataCache.public.clusterUUID**

строка

- **status.metadataCache.public.rootCA**
строка
- **status.metadataCache.publicLastFetchTimestamp**
строка

IstioMulticluster

Scope: Cluster

Version: v1alpha1

Ресурс для обозначения удалённого кластера как доверенного.

- **spec**
объект

Обязательный параметр

Доступно в редакциях: EE, CSE Pro (1.73)

- **spec.enableIngressGateway**
булевый

Доступно в редакциях: EE, CSE Pro (1.73)

Использование IngressGateway для доступа к удалённым подам.

Если поды между кластерами могут быть напрямую связаны друг с другом (например, при использовании «плоской» сети — flat network), тогда использование дополнительного маршрута может быть неэффективным, и лучше избежать лишних переходов.

По умолчанию: `true`

- **spec.metadata**
объект
 - **spec.metadata.ca**
строка

Доступно в редакциях: EE, CSE Pro (1.73)

HTTPS-сертификат удостоверяющего центра для чтения метаданных удалённого кластера.

- **spec.metadata.insecureSkipVerify**
булевый

Доступно в редакциях: EE, CSE Pro (1.73)

Включение или выключение проверки сертификата удостоверяющего центра для чтения метаданных удалённого кластера.

По умолчанию: `false`

Пример:

```
insecureSkipVerify: true
```

- **spec.metadataEndpoint**
строка

Обязательный параметр

Доступно в редакциях: EE, CSE Pro (1.73)

HTTPS-эндпоинт с метаданными удалённого кластера.

Пример:

```
metadataEndpoint: https://istio.k8s.example.com/metadata/
```

- **status**
объект
 - **status.metadataCache**
объект
 - **status.metadataCache.private**
объект
 - **status.metadataCache.private.apiHost**
строка
 - **status.metadataCache.private.ca**
строка
 - **status.metadataCache.private.ingressGateways**
массив объектов
 - **status.metadataCache.private.ingressGateways.address**
строка
 - **status.metadataCache.private.ingressGateways.port**
целочисленный
 - **status.metadataCache.private.networkName**
строка
 - **status.metadataCache.privateLastFetchTimestamp**

- строка
- **status.metadataCache.public**
- объект
 - **status.metadataCache.public.authnKeyPub**
 - строка
 - **status.metadataCache.public.clusterUUID**
 - строка
 - **status.metadataCache.public.rootCA**
 - строка
- **status.metadataCache.publicLastFetchTimestamp**
- строка

Кастомные ресурсы (от istio.io)

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Маршрутизация

DestinationRule

Reference

Позволяет:

- Определить стратегию балансировки трафика между эндпоинтами сервиса:
 - алгоритм балансировки (LEAST_CONN, ROUND_ROBIN, ...);
 - признаки смерти эндпоинта и правила его выведения из балансировки;
 - лимиты TCP-соединений и запросы для эндпоинтов;
 - Sticky Sessions;
 - Circuit Breaker.
- Определить альтернативные группы эндпоинтов для обработки трафика (применимо для Canary Deployments). При этом у каждой группы можно настроить свои стратегии балансировки.
- Настройка TLS для исходящих запросов.

VirtualService

Reference

Использование VirtualService опционально, классические сервисы продолжают работать, если вам достаточно их функционала.

Позволяет настроить маршрутизацию запросов:

- Аргументы для принятия решения о маршруте:
 - Host;
 - URI;
 - Weight.
- Параметры итоговых направлений:
 - новый хост;
 - новый URI;
 - если хост определен с помощью DestinationRule, можно направлять запросы на `subset`;
 - настройка времени ожидания и количества повторных запросов.



Istio необходимо знать о существовании `destination`, если вы используете внешний API, то зарегистрируйте его через ServiceEntry.

ServiceEntry

Reference

Аналог Endpoints + Service из ванильного Kubernetes. Позволяет сообщить Istio о существовании внешнего сервиса или даже переопределить его адрес.

Аутентификация

Решает задачу «Кто сделал запрос?». Не путать с авторизацией, которая определяет, «разрешить ли аутентифицированному элементу делать что-то или нет».

По факту есть два метода аутентификации:

- mTLS;
- JWT-токены.

PeerAuthentication

Reference

Позволяет определить стратегию mTLS в отдельном неймспейсе — принимать или нет нешифрованные запросы. Каждый mTLS-запрос автоматически позволяет определить источник и использовать его в правилах авторизации.

RequestAuthentication

Reference

Позволяет настроить JWT-аутентификацию для запросов.

Авторизация



Авторизация без mTLS- или JWT-аутентификации не будет работать в полной мере. В этом случае будут доступны только простейшие аргументы для составления политик, такие как `source.ip` и `request.headers`.

AuthorizationPolicy

Reference.

Включает и определяет контроль доступа к `workload`. Поддерживает как ALLOW-, так и DENY-правила. Как только у `workload` появляется хотя бы одна политика, начинает работать следующий приоритет:

- Если запрос попадает под политику DENY — запретить запрос.
- Если для данного приложения нет политик ALLOW — разрешить запрос.
- Если запрос попадает под политику ALLOW — разрешить запрос.
- Все остальные запросы — запретить.

Аргументы для принятия решения об авторизации:

- `source`:
 - `namespace`;
 - `principal` (идентификатор пользователя, полученный после аутентификации);
 - `IP`.
- `destination`:
 - метод (GET, POST...);
 - `Host`;
 - порт;
 - `URI`.
- `conditions`:
 - HTTP-заголовки
 - аргументы `source`
 - аргументы `destination`
 - JWT-токены

Sidecar

Reference

Данный ресурс позволяет ограничить количество сервисов, информация о которых будет передана в сайдкар `istio-proxy`.

Примеры

Circuit Breaker

Для выявления проблемных эндпоинтов используются настройки `outlierDetection` в кастомном ресурсе `DestinationRule`. Более подробно алгоритм Outlier Detection описан в документации Envoy.

Пример:


```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: reviews-cb-policy
spec:
  host: reviews.prod.svc.cluster.local
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100 # Максимальное число соединений в сторону host, суммарно для всех
        эндпоинтов.
      http:
        maxRequestsPerConnection: 10 # Каждые 10 запросов, соединение будет пересоздаваться.
    outlierDetection:
      consecutive5xxErrors: 7 # Допустимо 7 ошибок (включая пятисотые, TCP-таймауты и HTTP-таймауты)
      interval: 5m # в течение пяти минут,
      baseEjectionTime: 15m # после которых эндпоинт будет исключен из балансировки на 15 минут.

```

А также для настройки HTTP-таймаутов используется ресурс `VirtualService`. Эти таймауты также учитываются при подсчете статистики ошибок на эндпоинтах.

Пример:

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: my-productpage-rule
  namespace: myns
spec:
  hosts:
    - productpage
  http:
    - timeout: 5s
      route:
        - destination:
            host: productpage

```

Балансировка gRPC

⚠ Чтобы балансировка gRPC-сервисов заработала автоматически, присвойте имя с префиксом или значением `grpc` для порта в соответствующем сервисе.

Locality Failover

i При необходимости ознакомьтесь с основной документацией.

Istio позволяет настроить приоритетный географический failover между эндпоинтами. Для определения зоны Istio использует лейблы узлов с соответствующей иерархией:

- `topology.istio.io/subzone`;
- `topology.kubernetes.io/zone`;
- `topology.kubernetes.io/region`.

Это полезно для межкластерного failover при использовании совместно с мультикластером.

⚠ Для включения Locality Failover используется ресурс `DestinationRule`, в котором также необходимо настроить `outlierDetection`.

Пример:

```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
spec:
  host: helloworld
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true # Включили Locality Failover.
    outlierDetection: # outlierDetection включить обязательно.
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m

```

Retry

С помощью ресурса VirtualService можно настроить Retry для запросов.

⚠ По умолчанию при возникновении ошибок все запросы (включая POST-запросы) выполняются повторно до трех раз.

Пример:

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: ratings-route
spec:
  hosts:
  - ratings.prod.svc.cluster.local
  http:
  - route:
    - destination:
        host: ratings.prod.svc.cluster.local
    retries:
      attempts: 3
      perTryTimeout: 2s
      retryOn: gateway-error,connect-failure,refused-stream

```

Canary

⚠ Istio отвечает лишь за гибкую маршрутизацию запросов, которая опирается на спецзаголовки запросов (например, cookie) или просто на случайность. За настройку этой маршрутизации и «переключение» между канареечными версиями отвечает CI/CD-система.

Подразумевается, что в одном неймспейсе развёрнуты два Deployment с разными версиями приложения. У подов разных версий разные лейблы (`version: v1` и `version: v2`).

Требуется настроить два кастомных ресурса:

- DestinationRule с описанием, как идентифицировать разные версии вашего приложения (subset);
- VirtualService с описанием, как распределять трафик между разными версиями приложения.

Пример:

```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: productpage-canary
spec:
  host: productpage
  # subset доступны только при обращении к хосту через VirtualService из пода под управлением
  Istio.
  # Эти subset должны быть указаны в маршрутах.
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2

```

Распределение по наличию cookie

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - match:
    - headers:
        cookie:
          regex: "^(.*;)?(canary=yes)(;.*)?"
    route:
    - destination:
        host: productpage
        subset: v2 # Ссылка на subset из DestinationRule.
    - route:
        - destination:
            host: productpage
            subset: v1

```

Распределение по вероятности

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1 # Ссылка на subset из DestinationRule.
        weight: 90 # Процент трафика, который получают поды с лейблом version: v1.
    - route:
        - destination:
            host: productpage
            subset: v2
            weight: 10

```

Ingress для публикации приложений

Istio Ingress Gateway

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
  name: main
spec:
  # ingressGatewayClass содержит значение селектора лейблов, используемое при создании ресурса
  Gateway.
  ingressGatewayClass: istio-hp
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  resourcesRequests:
    mode: VPA

```

```

apiVersion: v1
kind: Secret
metadata:
  name: app-tls-secret
  namespace: d8-ingress-istio # Обратите внимание, неймспейс не является app-ns.
type: kubernetes.io/tls
data:
  tls.crt: |
    <tls.crt data>
  tls.key: |
    <tls.key data>

```

```

apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway-app
  namespace: app-ns
spec:
  selector:
    # Селектор лейблов для использования Istio Ingress Gateway main-hp.
    istio.deckhouse.io/ingress-gateway-class: istio-hp
  servers:
    - port:
        # Стандартный шаблон для использования протокола HTTP.
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - app.example.com
    - port:
        # Стандартный шаблон для использования протокола HTTPS.
        number: 443
        name: https
        protocol: HTTPS
      tls:
        mode: SIMPLE
        # Ресурс Secret с сертификатом и ключом, который должен быть создан в неймспейсе d8-ingress-istio.
        # Поддерживаемые форматы ресурсов Secret можно посмотреть по ссылке
        https://istio.io/latest/docs/tasks/traffic-management/ingress/secure-ingress/#key-formats.
        credentialName: app-tls-secret
      hosts:
        - app.example.com

```

```

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            host: app-svc

```

Ingress NGINX

Для работы с Ingress NGINX требуется подготовить:

- Ingress-контроллер, добавив к нему сайдкар от Istio. В нашем случае — включить параметр `enableIstioSidecar` в кастомном ресурсе `IngressNginxController` модуля `ingress-nginx`.
- Ingress-ресурс, который ссылается на Service. Обязательные аннотации для Ingress-ресурса:
 - `nginx.ingress.kubernetes.io/service-upstream: "true"` — с этой аннотацией Ingress-контроллер будет отправлять запросы на ClusterIP сервиса (из диапазона Service CIDR) вместо того, чтобы слать их напрямую в поды приложения. Sidecar-контейнер `istio-proxy` перехватывает трафик только в сторону диапазона Service CIDR, остальные запросы отправляются напрямую;
 - `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — с данной аннотацией сайдкар сможет идентифицировать прикладной сервис, для которого предназначен запрос.

Примеры:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Просим nginx проксировать трафик на ClusterIP вместо собственных IP подов.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # В Istio вся маршрутизация осуществляется на основе `Host:` заголовка запросов.
    # Чтобы не сообщать Istio о существовании внешнего домена `productpage.example.com`,
    # мы просто используем внутренний домен, о котором Istio осведомлен.
    nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc
spec:
  rules:
    - host: productpage.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
          backend:
            service:
              name: productpage
              port:
                number: 9080

```

```

apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
    - name: http
      port: 9080
  selector:
    app: productpage
  type: ClusterIP

```

Примеры настройки авторизации

Алгоритм принятия решения

Как только для приложения создается AuthorizationPolicy, начинает работать следующий алгоритм принятия решения о судьбе запроса:

- Если запрос попадает под политику DENY — запретить запрос.
- Если для данного приложения нет политик ALLOW — разрешить запрос.
- Если запрос попадает под политику ALLOW — разрешить запрос.
- Все остальные запросы — запретить.

Иными словами, если вы явно что-то запретили, работает только ваш запрет. Если же вы что-то явно разрешили, теперь разрешены только явно одобренные запросы (запреты никуда не исчезают и имеют приоритет).



Для работы политик, основанных на высокоуровневых параметрах, таких как неймспейсы или principal, необходимо, чтобы все вовлеченные сервисы работали под управлением Istio. Также между приложениями должен быть организован Mutual TLS.

Примеры:

- Запретим POST-запросы для приложения myapp. Отныне, так как для приложения появилась политика, согласно алгоритму выше будут запрещены только POST-запросы к приложению.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-post-requests
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: DENY
  rules:
  - to:
    - operation:
        methods: ["POST"]
```

- Для приложения создана разрешающая политика ALLOW. При ней будут разрешены только запросы из неймспейса bar, остальные запрещены.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # default, можно не указывать.
  rules:
  - from:
    - source:
        namespaces: ["bar"]
```

- Здесь для приложения создана политика ALLOW. При этом она не содержит ни одного правила, вследствие чего ни один запрос не попадает под критерии фильтрации. Однако сам факт наличия политики активирует алгоритм проверки, где всё, что не разрешено явно, считается запрещенным. В данном случае блокируются абсолютно все запросы

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # default, можно не указывать.
  rules: []
```

- Здесь для приложения создана политика ALLOW (это default) и одно пустое правило. Под это правило попадает любой запрос и автоматически получает разрешение.

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  rules:
  - {}

```

Запретить все действия в рамках неймспейса foo

Два способа:

- Запретить явно. Здесь мы создаем политику DENY с единственным универсальным фильтром {}, под который попадают все запросы:

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  action: DENY
  rules:
  - {}

```

- Неявно. Здесь мы создаем политику ALLOW (по умолчанию), но не создаем ни одного фильтра, так что ни один запрос под нее не попадет и будет автоматически запрещен.

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec: {}

```

Запретить доступ только из неймспейса foo

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-from-ns-foo
  namespace: myns
spec:
  action: DENY
  rules:
  - from:
    - source:
        namespaces: ["foo"]

```

Разрешить запросы только в рамках нашего неймспейса foo

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-intra-namespace-only
  namespace: foo
spec:
  action: ALLOW
  rules:
  - from:
    - source:
        namespaces: ["foo"]

```

Разрешить из любого места в нашем кластере

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all-from-my-cluster
  namespace: myns
spec:
  action: ALLOW
  rules:
  - from:
    - source:
      principals: ["mycluster.local/*"]

```

Разрешить любые запросы только кластеров foo или bar

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all-from-foo-or-bar-clusters-to-ns-baz
  namespace: baz
spec:
  action: ALLOW
  rules:
  - from:
    - source:
      principals: ["foo.local/*", "bar.local/*"]

```

Разрешить любые запросы только от сущностей из неймспейса baz кластеров foo или bar

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all-from-foo-or-bar-clusters-to-ns-baz
  namespace: baz
spec:
  action: ALLOW
  rules:
  - from:
    - source: # Правила ниже логически перемножаются.
      namespaces: ["baz"]
      principals: ["foo.local/*", "bar.local/*"]

```

Разрешить из любого кластера (по mTLS)



Если есть запрещающие правила, у них будет приоритет. Подробнее можно посмотреть алгоритм.

Пример:

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all-from-any-cluster-with-mtls
  namespace: myns
spec:
  action: ALLOW
  rules:
  - from:
    - source:
      principals: ["*"] # To set mTLS mandatory.

```

Разрешить вообще откуда угодно (в том числе без mTLS)


```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all-from-any
  namespace: myns
spec:
  action: ALLOW
  rules: [{}]
```

Устройство федерации из двух кластеров с помощью кастомного ресурса IstioFederation

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Устройство мультикластера из двух кластеров с помощью ресурса IstioMulticluste

 Доступно только в редакции Enterprise Edition.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluste
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluste
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Управление поведением data plane

Предотвращение завершения работы istio-proxy до завершения соединений основного приложения

По умолчанию в процессе остановки пода все контейнеры, включая istio-proxy, получают сигнал SIGTERM одновременно. Но некоторым приложениям для правильного завершения работы необходимо время и иногда дополнительная сетевая активность. Это невозможно, если istio-proxy завершился раньше.

Решение — добавить в istio-proxy preStop-хук для оценки активности прикладных контейнеров, а единственный доступный метод — это выявление сетевых сокетов приложения, и если таковых нет, тогда можно останавливать контейнер.

Аннотация ниже добавляет описанный выше preStop-хук в контейнер istio-proxy прикладного пода:

```
annotations:
  inject.istio.io/templates: "sidecar,d8-hold-istio-proxy-termination-until-application-stops"
```

Ограничения режима перенаправления прикладного трафика CNIPlugin

В отличие от режима `InitContainer`, настройка перенаправления осуществляется в момент создания пода, а не в момент срабатывания `init`-контейнера `istio-init`. Это значит, что прикладные `init`-контейнеры не смогут взаимодействовать с остальными сервисами так как весь трафик будет перенаправлен на обработку в `sidecar`-контейнер `istio-proxy`, который ещё не запущен. Обходные пути:

- Запустить прикладной `init`-контейнер от пользователя с `uid` `1337`. Запросы данного пользователя не перехватываются под управление Istio.
- Исключить IP-адрес или порт сервиса из-под контроля Istio с помощью аннотаций `traffic.sidecar.istio.io/excludeOutboundIPRanges` или `traffic.sidecar.istio.io/excludeOutboundPorts`.

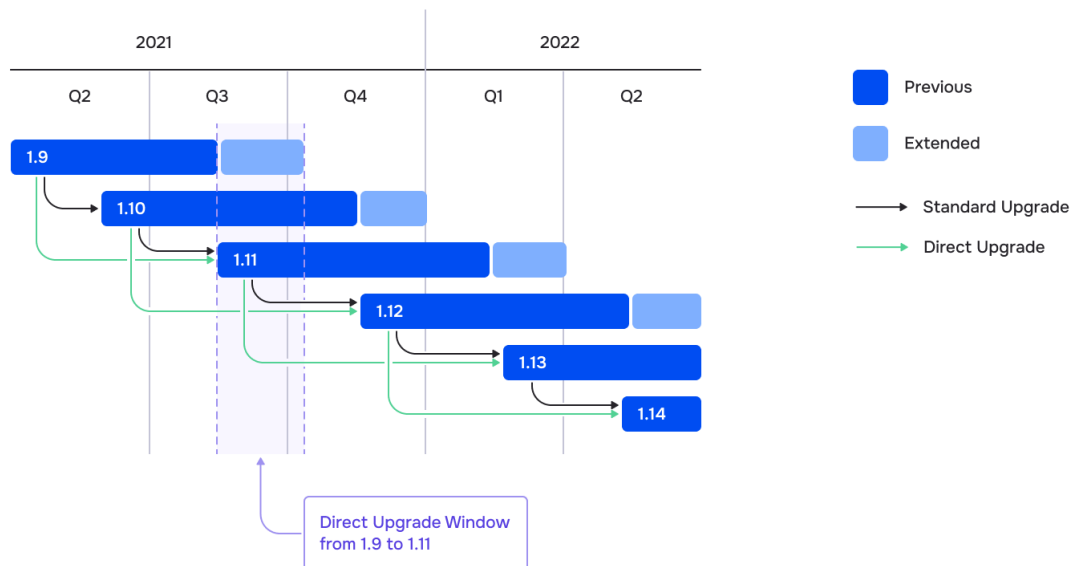


Каждый из обходных вариантов выводит трафик из-под контроля Istio, что в свою очередь убирает шифрование трафика между прикладными сервисами.

Обновление Istio

Обновление control plane Istio

- Deckhouse Kubernetes Platform позволяет установить несколько версий control plane одновременно:
 - Одна глобальная, обслуживает неймспейсы или поды без явного указания версии (лейбл у неймспейсов `istio-injection: enabled`). Настраивается параметром `globalVersion`.
 - Остальные — дополнительные, обслуживают неймспейсы или поды с явным указанием версии (лейбл у неймспейса или пода `istio.io/rev: v1x21`). Настраиваются параметром `additionalVersions`.
- Istio заявляет обратную совместимость между data plane и control plane в диапазоне двух минорных версий:



- Алгоритм обновления (для примера, с версии `1.21` на версию `1.25`):
 - Добавить желаемую версию в параметр модуля `additionalVersions` (`additionalVersions: ["1.25"]`).
 - Дождаться появления соответствующего пода `istiod-v1x25-xxx-yyy` в неймспейсе `d8-istio`.
 - Для каждого прикладного неймспейса, где включен Istio:
 - поменять лейбл `istio-injection: enabled` на `istio.io/rev: v1x25`;
 - по очереди пересоздать поды в неймспейсе, параллельно контролируя работоспособность приложения.
 - Поменять настройку `globalVersion` на `1.25` и удалить `additionalVersions`.
 - Убедиться, что старый под `istiod` удален.
 - Поменять лейблы прикладных неймспейсов на `istio-injection: enabled`.

Чтобы найти все поды под управлением старой ревизии Istio (в примере — версия 21), выполните команду:

```
d8 k get pods -A -o json | jq --arg revision "v1x21" \
  '.items[] | select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
    .revision == $revision) | .metadata.namespace + "/" + .metadata.name'
```



Обновление до версии Istio 1.25 возможно только с версии 1.21.

Автоматическое обновление data plane Istio

Для автоматизации обновления `istio`-сайдкаров установите лейбл `istio.deckhouse.io/auto-upgrade="true"` на Namespace либо на отдельный ресурс — `Deployment`, `DaemonSet` или `StatefulSet`.

Настройка ресурсов istio-proxy sidecar

Для переопределения глобальных ограничений ресурсов для сайдкара istio-проху в отдельных рабочих нагрузках используются аннотации. Поддерживаются следующие аннотации:

Поддерживаемые аннотации

Аннотация	Описание	Пример значения
sidecar.istio.io/proxyCPU	Запрос CPU для сайдкара	200m
sidecar.istio.io/proxyCPULimit	Лимит CPU для сайдкара	"1"
sidecar.istio.io/proxyMemory	Запрос памяти для сайдкара	128Mi
sidecar.istio.io/proxyMemoryLimit	Лимит памяти для сайдкара	512Mi

Примеры конфигурации

Для Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/proxyCPU: 200m
        sidecar.istio.io/proxyCPULimit: "1"
        sidecar.istio.io/proxyMemory: 128Mi
        sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... остальная часть манифеста
```

Для ReplicaSet:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/proxyCPU: 200m
        sidecar.istio.io/proxyCPULimit: "1"
        sidecar.istio.io/proxyMemory: 128Mi
        sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... остальная часть манифеста
```

Для Pod:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    sidecar.istio.io/proxyCPU: 200m
    sidecar.istio.io/proxyCPULimit: "1"
    sidecar.istio.io/proxyMemory: 128Mi
    sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... остальная часть манифеста
```

 Все четыре параметра должны быть указаны вместе — sidecar.istio.io/proxyCPU, sidecar.istio.io/proxyCPULimit, sidecar.istio.io/proxyMemory и sidecar.istio.io/proxyMemoryLimit. Частичная конфигурация не поддерживается.

Модуль keepalived

⚠ Данный модуль не работает с модулем cilium.

Модуль с управлением через custom resources для настройки keepalived-кластеров на узлах.

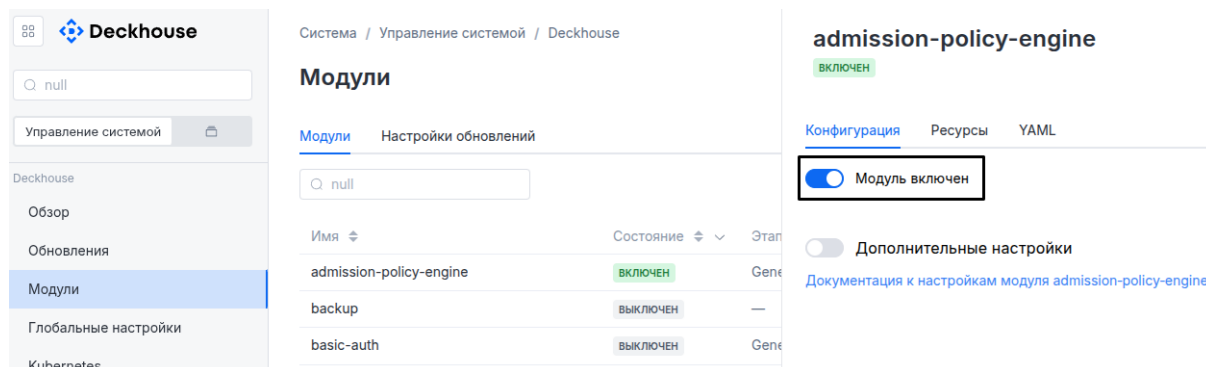
Настройки

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `keepalived`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `keepalived`:

```
d8 system module enable keepalived
```

- **С помощью ModuleConfig `keepalived`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `keepalived` (создайте его, при необходимости).

Пример манифеста для включения модуля `keepalived`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: keepalived
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `keepalived`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `keepalived` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `keepalived`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `keepalived` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc keepalived
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `keepalived`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `keepalived`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: keepalived
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Настройка `keepalived`-кластеров выполняется с помощью `custom resource`.

Custom resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

KeepalivedInstance

Scope: Cluster
Version: v1alpha1

Custom resource для настройки `keepalived`-кластеров.

Один custom resource `KeepalivedInstance` описывает один `keepalived`-кластер.

`metadata.name` — имя кластера, используется в названиях подов.

- **spec**
объект
Обязательный параметр
 - **spec.nodeSelector**
объект

Обязательный параметр

Определяет `nodeSelector` для подов с keepalived.

Формат: стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- о **spec.tolerations**

массив объектов

Определяет `tolerations` для подов с keepalived.

Формат: стандартный список `toleration`. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**

строка

Определяет, какому `effect`'у `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому `effect`.

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**

строка

Определяет, какому ключу (`key`) `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому ключу.

Если ключ (`key`) не указан (пуст), `operator` должен быть `Exists`, что подразумевает соответствие любым `value` и `key`.

- **spec.tolerations.operator**

строка

Определяет отношение ключа (`key`) к значению (`value`) — должен ли ключ быть равен (`Equal`) значению или допустимо существование ключа с любым значением.

Указание `Exists` означает, что значение `value` может быть любым, чтобы под с указанным `toleration` был размещён на узле с соответствующим `taint`.

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**

целочисленный

Определяет период времени в секундах, в течение которого планировщик должен ждать, прежде чем выселить (`evict`) под с узла, если `toleration` больше не соответствует `taint`'у (справедливо **только** для `effect` `NoExecute`, иначе игнорируется).

Если **не установлено**, тогда под не будет выселен с узла, если `toleration` больше не соответствует `taint`. Если **установлен в ноль** (или отрицателен), под будет выселен с узла немедленно, если `toleration` больше не соответствует `taint`.

По умолчанию используется значение «не установлено».

- **spec.tolerations.value**

строка

Значение, которому должен соответствовать `toleration`.

Должно быть пустым, если `operator` имеет параметр `Exists`.

- о **spec.vrrpInstances**

массив объектов

Обязательный параметр

Список инстансов VRRP внутри keepalived-кластера. По сути, список групп адресов, которые мигрируют между серверами одновременно и не могут работать друг без друга.

Не нужно путать `vrrpInstance` и `KeepalivedInstance`. `vrrpInstance` — это составная часть `KeepalivedInstance`.

Модуль `keepalived` настраивает VRRP-инстансы таким образом, чтобы все адреса (все группы) не собирались одновременно на одном узле, а распределялись равномерно по всем серверам.

- **spec.vrrpInstances.id**

целочисленный

Обязательный параметр

Уникальный в масштабах всего кластера идентификатор VRRP-группы.

Не нужно использовать одинаковый ID в разных инстансах `KeepalivedInstance`, если у вас на это нет особой причины.

Допустимые значения: `1 <= X <= 255`

- **spec.vrrpInstances.interface**

объект

Обязательный параметр

Определяет, как вычислить интерфейс для служебного VRRP-трафика на узле.

■ **spec.vrrpInstances.interface.detectionStrategy**

строка

Одна из трех возможных стратегий определения интерфейса для трафика VRRP:

- **Name** — задать имя интерфейса явно, с помощью параметра `spec.vrrpInstances[].interface.name`. В этом случае все узлы должны иметь одинаковый интерфейс, который смотрит в нужную сеть (например, eth0);
- **NetworkAddress** — найти на узле интерфейс с IP из этой подсети и использовать его;
- **DefaultRoute** — использовать интерфейс, через который определен маршрут по умолчанию на узле (в таблице 254 «main»).

■ **spec.vrrpInstances.interface.name**

строка

Имя интерфейса для служебного VRRP-трафика в случае использования `detectionStrategy = Name`.

Обязательный параметр в случае использования `detectionStrategy = Name`.

■ **spec.vrrpInstances.interface.networkAddress**

строка

Интерфейс узла с IP-адресом из этой подсети будет использован как служебный в случае использования `detectionStrategy = NetworkAddress`.

Обязательный параметр в случае использования `detectionStrategy = NetworkAddress`.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Пример:

```
networkAddress: 192.168.42.0/24
```

■ **spec.vrrpInstances.preempt**

булевый

Возвращать ли IP на узел, который восстановился после аварии.

Если у вас один `vrrpInstance`, разумнее не перекидывать IP лишний раз, чтобы не трогать соединения. Если групп много и трафик большой, лучше вернуть, чтобы не допустить скопления всех групп на одном узле.

По умолчанию используется значение `true`, то есть IP вернется на узел в случае, если он вернется в строй.

■ **spec.vrrpInstances.virtualIPAddresses**

массив объектов

Обязательный параметр

Список IP-адресов, которые **одновременно** будут «прыгать» между серверами.

■ **spec.vrrpInstances.virtualIPAddresses.address**

строка

Обязательный параметр

Один из адресов в группе.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Пример:

```
address: 192.168.42.15/32
```

■ **spec.vrrpInstances.virtualIPAddresses.interface**

объект

Необязательный параметр.

Интерфейс для привязки виртуального IP-адреса на узле. Аналогичен `spec.vrrpInstances[].interface`.

Если не указать — будет использован основной, служебный интерфейс, который определен в `spec.vrrpInstances[].interface`.

■ **spec.vrrpInstances.virtualIPAddresses.interface.detectionStrategy**

строка

Одна из трех возможных стратегий определения интерфейса для привязки IP-адреса:

- **Name** — задать имя интерфейса явно, с помощью параметра `spec.vrrpInstances[].virtualIPAddresses[].interface.name`. В этом случае все узлы должны иметь одинаковый интерфейс, который смотрит в нужную сеть (например, eth0);

- **NetworkAddress** — найти на узле интерфейс с IP из этой подсети и использовать его;
- **DefaultRoute** — использовать интерфейс, через который определен маршрут по умолчанию на узле (в таблице 254 «main»).
- **spec.vrrpInstances.virtualIPAddresses.interface.name**
строка

Имя интерфейса для привязки виртуального IP-адреса в случае использования `virtualIPAddresses[].detectionStrategy = Name`.

Обязательный параметр в случае использования `virtualIPAddresses[].detectionStrategy = Name`.

- **spec.vrrpInstances.virtualIPAddresses.interface.networkAddress**
строка

Интерфейс узла с IP-адресом из этой подсети будет использован как служебный в случае использования `virtualIPAddresses[].detectionStrategy = NetworkAddress`.

Обязательный параметр в случае использования `virtualIPAddresses[].detectionStrategy = NetworkAddress`.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Пример:

```
networkAddress: 192.168.42.0/24
```

keepalived: FAQ

Как вручную переключить keepalived?

1. Зайдите в нужный под, используя debug-контейнер с общим пространством процессов: `d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived`.
2. Отредактируйте файл конфигурации `vim /proc/1/root/etc/keepalived/keepalived.conf`, где в строке с параметром `priority` замените значение на <число подов keepalived + 1> или установите значение выше, чем у текущего VRRP-мастера (например, 255).
3. Примените настройки – отправьте сигнал на перечитывание конфигурации: `kill -HUP 1`.

Примеры

Три публичных IP-адреса

Три публичных IP-адреса на трех front-узлах. Каждый виртуальный IP-адрес вынесен в отдельную VRRP-группу, таким образом, каждый адрес «прыгает» независимо от других, и если в кластере три узла с лейблами `node-role.deckhouse.io/frontend: ""`, то каждый IP получит по своему master-узлу.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Обязательно.
    node-role.deckhouse.io/frontend: ""
  tolerations: # Опционально.
    - key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  vrrpInstances:
    - id: 1 # Уникальный для всего кластера ID.
      interface:
        detectionStrategy: DefaultRoute # В качестве служебной сетевой карты используем ту, через
        которую проложен дефолтный маршрут.
        virtualIPAddresses:
          - address: 42.43.44.101/32
            # В нашем примере адреса «прыгают» по тем же сетевым интерфейсам, по которым ходит служебный
            VRRP-трафик, поэтому мы не указываем параметр interface.
    - id: 2
      interface:
        detectionStrategy: DefaultRoute
        virtualIPAddresses:
          - address: 42.43.44.102/32
    - id: 3
      interface:
        detectionStrategy: DefaultRoute
        virtualIPAddresses:
          - address: 42.43.44.103/32
```


Шлюз с парой IP-адресов для LAN и WAN. В случае шлюза приватный и публичный IP друг без друга не могут и «прыгать» между узлами они будут вместе. Служебный VRRP-трафик в данном примере мы решили пустить через LAN-интерфейс, который мы определим с помощью метода NetworkAddress (считаем, что на каждом узле есть IP из этой подсети).

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
    - key: node-role.deckhouse.io/mygateway
      operator: Exists
  vrrpInstances:
    - id: 4 # ID "1", "2", "3" уже заняты в KeepalivedInstance "front" выше.
      interface:
        detectionStrategy: NetworkAddress
        networkAddress: 192.168.42.0/24
      virtualIPAddresses:
        - address: 192.168.42.1/24
          # В данном случае мы уже определили локальную сеть выше и можем не определять интерфейс для
          # этого IP, не указав параметр interface.
        - address: 42.43.44.1/28
          interface:
            detectionStrategy: Name
            name: ens7 # На всех узлах интерфейс для публичных IP называется "ens7", воспользуемся этим.
```

Модуль kube-dns

Модуль устанавливает компоненты CoreDNS для управления DNS в кластере Kubernetes.

Внимание. Модуль удаляет ранее установленные kubeadm'ом Deployment, ConfigMap и RBAC для CoreDNS. При разворачивании собственного CoreDNS избегайте использования имен `coredns` или `system:coredns` для любых ресурсов (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Используйте альтернативные имена, например `infra-dns`, чтобы предотвратить их автоматическое удаление Deckhouse Kubernetes Platform.

Настройки

В модуле реализован 1 алерт.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `kube-dns`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `kube-dns`:

```
d8 system module enable kube-dns
```

- **С помощью ModuleConfig kube-dns.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `kube-dns` (создайте его, при необходимости).

Пример манифеста для включения модуля `kube-dns`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `kube-dns`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `kube-dns` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `kube-dns`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `kube-dns` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc kube-dns
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `kube-dns`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `kube-dns`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.clusterDomainAliases**

массив строк

Список альтернативных доменов кластера, разрешаемых наравне с `global.discovery.clusterDomain`.

Обратите внимание: альтернативный домен не должен совпадать с доменом, используемым в шаблоне DNS-имен в параметре `publicDomainTemplate`.

- **Элемент массива**

строка

Шаблон: `^[0-9a-zA-Z\.-]+$`

- **settings.enableLogs**

булевый

Включение логирования в CoreDNS.

По умолчанию: `false`

- **settings.hosts**

массив объектов

Необязательный параметр.

Статический список хостов в стиле `/etc/hosts`.

- **settings.hosts.domain**

строка

Шаблон: `^[0-9a-zA-Z\.-]+$`

- **settings.hosts.ip**

строка

Шаблон: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}$`

- **settings.stubZones**

массив объектов

Список дополнительных зон для обслуживания CoreDNS.

По умолчанию: `[]`

Пример:

```

stubZones:
- zone: example.com
  upstreamNameservers:
  - 8.8.8.8
  cacheTTLSeconds: 3600
  transportProtocolMode: PreferUDP

```

- **settings.stubZones.cacheTTLSeconds**

целочисленный

Максимальный TTL в секундах для успешных запросов.

По умолчанию: `30`

Допустимые значения: `1 <= X <= 3600`

- **settings.stubZones.transportProtocolMode**

строка

Режим транспортного протокола взаимодействия с вышестоящим DNS-сервером:

- `PreferUDP` — Используется UDP, даже если первоначальный запрос пришел по TCP. Обратите внимание, что если DNS-ответ от вышестоящего DNS-сервера превысит 512 байт, был усечен и помечен битом TC (RFC 1035, то он будет повторно отправлен по TCP (RFC 5966). В случае

- каких-либо других проблем с UDP, запрос повторно по TCP отправлен не будет.
- **ForceTCP** — Всегда используется TCP, даже если исходный запрос пришел по UDP.
- **KeepOriginal** — Используется тот же протокол (TCP или UDP), что и в исходном запросе.

По умолчанию: **PreferUDP**

Допустимые значения: **PreferUDP**, **ForceTCP**, **KeepOriginal**

■ **settings.stubZones.upstreamNameservers**

массив строк

Обязательный параметр

Список IP-адресов рекурсивных DNS-серверов, которые CoreDNS будет использовать для разрешения доменов в этой зоне.

■ **Элемент массива**

строка

Шаблон: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?&`

■ **settings.stubZones.zone**

строка

Обязательный параметр

Зона CoreDNS.

Шаблон: `^[0-9a-zA-Z\.\- _]+&`

Пример:

```
zone: consul.local
```

○ **settings.transportProtocolMode**

строка

Режим транспортного протокола взаимодействия с вышестоящим DNS-сервером:

- **PreferUDP** — Используется UDP, даже если первоначальный запрос пришел по TCP. Обратите внимание, что если DNS-ответ от вышестоящего DNS-сервера превысит 512 байт, был усечен и помечен битом TC (RFC 1035, то он будет повторно отправлен по TCP (RFC 5966). В случае каких-либо других проблем с UDP, запрос повторно по TCP отправлен не будет.
- **ForceTCP** — Всегда используется TCP, даже если исходный запрос пришел по UDP.
- **KeepOriginal** — Используется тот же протокол (TCP или UDP), что и в исходном запросе.

По умолчанию: **PreferUDP**

Допустимые значения: **PreferUDP**, **ForceTCP**, **KeepOriginal**

○ **settings.upstreamNameservers**

массив строк

Список IP-адресов рекурсивных DNS-серверов, которые CoreDNS будет использовать для разрешения внешних доменов.

По умолчанию используется список из `/etc/resolv.conf`.

■ **Элемент массива**

строка

Шаблон: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?&`

FAQ

Как поменять домен кластера с минимальным простоем?

Добавьте новый домен и сохраните предыдущий. Для этого измените конфигурацию параметров:

1. В controlPlaneManager.apiserver:

- controlPlaneManager.apiserver.certSANS,
- apiserver.serviceAccount.additionalAPIAudiences,
- apiserver.serviceAccount.additionalAPIIssuers.

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  enabled: true
  settings:
    apiserver:
      certSANs:
        - kubernetes.default.svc.<старый clusterDomain>
        - kubernetes.default.svc.<новый clusterDomain>
      serviceAccount:
        additionalAPIAudiences:
          - https://kubernetes.default.svc.<старый clusterDomain>
          - https://kubernetes.default.svc.<новый clusterDomain>
        additionalAPIIssuers:
          - https://kubernetes.default.svc.<старый clusterDomain>
          - https://kubernetes.default.svc.<новый clusterDomain>

```

2. В kubeDns.clusterDomainAliases:

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    clusterDomainAliases:
      - <старый clusterDomain>
      - <новый clusterDomain>

```

3. Дождитесь перезапуска kube-apiserver:

```
d8 k -n kube-system get pods -l component=kube-apiserver
```

4. Поменяйте clusterDomain на новый. Для этого выполните команду:

```
d8 system edit cluster-configuration
```

5. Перезапустите поды deckhouse:

```
d8 k -n d8-system rollout restart deployment deckhouse
```



Важно. В Kubernetes, контроллеры используют расширенные токены для ServiceAccount для работы с API-server. Это означает, что каждый такой токен содержит дополнительные поля `iss:` и `aud:`, которые включают в себя старый `clusterDomain` (например, `"iss": "https://kubernetes.default.svc.cluster.local"`). При смене `clusterDomain` API-server начнет выдавать токены с новым `service-account-issuer`, но благодаря произведенной конфигурации `additionalAPIAudiences` и `additionalAPIIssuers` по-прежнему будет принимать старые токены. По истечении 48 минут (80% от 3607 секунд) Kubernetes начнет обновлять выпущенные токены, при обновлении будет использован новый `service-account-issuer`. Через 90 минут (3607 секунд плюс дополнительный буфер) после перезагрузки kube-apiserver можете удалить конфигурацию `serviceAccount` из конфигурации `control-plane-manager`.



Важно. Если необходимо убрать старый домен из `clusterDomainAliases` в конфигурации kube-dns, необходимо пересоздать все поды в кластере, чтобы они запустились с новым `search domain` в `/etc/resolv.conf`. Это приведет к недоступности сервисов кластера, пока поды не перезапустятся.

```
d8 k delete pods --all-namespaces --all
```



Важно. Если вы используете модуль `istio`, после смены `clusterDomain` обязательно потребуется рестарт всех прикладных подов под управлением Istio.

Как увеличить количество подов kube-dns?

Deckhouse распределяет поды kube-dns по следующему принципу: выполняется поиск узлов с лейблами `node-role.deckhouse.io/` и `node-role.kubernetes.io/`, затем применяются следующие правила:

- Если в кластере есть узлы с ролью `kube-dns`, количество реплик вычисляется как сумма таких узлов и master-узлов, но не больше чем количество master-узлов + 2.
- Если узлы kube-dns отсутствуют, производится поиск узлов с ролью `system`, и тогда количество реплик определяется как сумма system-узлов и master-узлов, но не больше чем количество master-узлов + 2.
- Если в кластере присутствуют только мастер-узлы, количество реплик kube-dns будет равно числу мастеров.

Примеры

Пример конфигурации

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
      - 8.8.8.8
      - 8.8.4.4
    hosts:
      - domain: one.example.com
        ip: 192.168.0.1
      - domain: two.another.example.com
        ip: 10.10.0.128
    stubZones:
      - zone: consul.local
        upstreamNameservers:
          - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
      - foo.bar
      - baz.qux
```

Модуль kube-proxy

Модуль удаляет весь комплект kube-proxy (`DaemonSet`, `ConfigMap`, `RBAC`) от `kubeadm` и устанавливает свой.

По умолчанию в целях безопасности при использовании сервисов с типом `NodePort` подключения принимаются только на `InternalIP` узлов. Для снятия данного ограничения предусмотрена аннотация на узел — `node.deckhouse.io/nodeport-bind-internal-ip: "false"`.

Пример аннотации для `NodeGroup`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
  ...
```

i **Внимание.** После добавления, удаления или изменения значения аннотации необходимо самостоятельно выполнить рестарт подов kube-proxy.

Внимание. Модуль kube-proxy автоматически отключается при включении модуля `cni-cilium`.

Модуль local-path-provisioner

Позволяет пользователям Kubernetes использовать локальное хранилище на узлах.

Как это работает?

Для каждого custom resource `LocalPathProvisioner` создается соответствующий `StorageClass`.

Допустимая топология для `StorageClass` вычисляется на основе списка `nodeGroup` из custom resource. Топология используется при шедулинге подов.

Когда под заказывает диск, то:

- создается `HostPath` PV;
- `Provisioner` создает на нужном узле локальную папку по пути, состоящем из параметра `path` custom resource, имени PV и имени PVC.

Пример пути:

```
/opt/local-path-provisioner/pvc-d9bd3878-f710-417b-a4b3-38811aa8aac1_d8-monitoring_prometheus-main-db-prometheus-main-0
```

Ограничения

- Ограничение на размер диска не поддерживается для локальных томов.

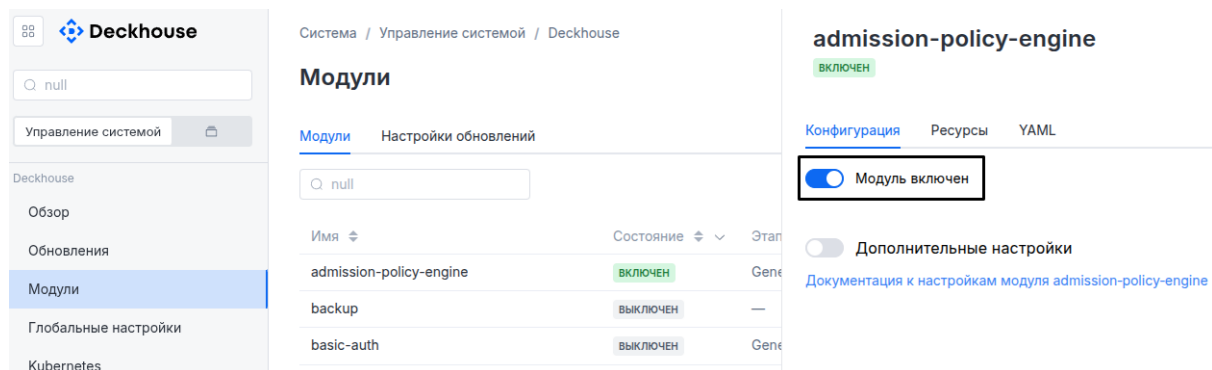
Настройки

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `local-path-provisioner`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `local-path-provisioner`:

```
d8 system module enable local-path-provisioner
```

- **С помощью ModuleConfig local-path-provisioner.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig local-path-provisioner` (создайте его, при необходимости).

Пример манифеста для включения модуля `local-path-provisioner`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: local-path-provisioner
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `local-path-provisioner`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `local-path-provisioner` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `local-path-provisioner`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `local-path-provisioner` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc local-path-provisioner
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `local-path-provisioner`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `local-path-provisioner`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: local-path-provisioner
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Модуль не требует конфигурации.

Custom resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

LocalPathProvisioner

Scope: Cluster

Version: v1alpha1

Параметры экземпляра `local-path-provisioner`.

- **spec**
объект
 - **spec.nodeGroups**
массив строк

nodeGroup'ы, в которых работает данный StorageClass.

Пример:


```
nodeGroups:
- master
- worker
```

◦ **спис.путь**

строка

Обязательный параметр

Корневой путь на узле к директории для хранения PV.

Пример:

```
path: "/opt/local-path-provisioner"
```

◦ **спис.рекламПолитика**

строка

Сохранять ли PV после удаления PVC.

По умолчанию: `Retain`

Допустимые значения: `Retain`, `Delete`

FAQ

Как настроить Prometheus на использование локального хранилища?

Применить custom resource `LocalPathProvisioner`:

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
спис:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

- `спис.нодГруппы` должен совпадать с NodeGroup, где запущен под Prometheus'a.
- `спис.путь` - путь на узле, где будут лежать данные.

Добавить в конфигурацию модуля `prometheus` следующие параметры:

```
longtermStorageClass: localpath-system
storageClass: localpath-system
```

Дождаться переката подов Prometheus.

Примеры

Пример custom resource `LocalPathProvisioner`

Reclaim policy устанавливается по умолчанию в `Retain`.

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
спис:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

Пример custom resource `LocalPathProvisioner` с установленным `reclaimPolicy`

Reclaim policy устанавливается в `Delete`.

```

apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
    - system
  path: "/opt/local-path-provisioner"
  reclaimPolicy: "Delete"

```

Модуль log-shipper

Модуль `log-shipper` упрощает настройку сбора логов в Kubernetes. Он позволяет быстро организовать сбор логов как с приложений, запущенных в кластере, так и с самих узлов, а затем отправлять их в любую систему хранения — внутреннюю или внешнюю (например, Loki, Elasticsearch, S3 и другие).

Возможности `log-shipper`:

- централизованно собирает и передаёт логи из кластера;
 - фильтрует, преобразовывает и обогащает логи перед отправкой;
 - настраивает маршрутизацию логов между различными источниками и приемниками.
1. Deckhouse Kubernetes Platform (DKP) следит за ресурсами `ClusterLoggingConfig`, `ClusterLogDestination` и `PodLoggingConfig`. Комбинация конфигурации для сбора логов и направления для отправки называется `pipeline`.
 2. DKP генерирует конфигурационный файл и сохраняет его в `Secret` в Kubernetes.
 3. `Secret` монтируется всем подам агентов `log-shipper`, конфигурация обновляется при ее изменении с помощью `sidecar`-контейнера `reloader`.

Топологии отправки

Этот модуль отвечает за агентов на каждом узле. Подразумевается, что логи из кластера отправляются согласно одной из описанных ниже топологий.

Распределенная

Агенты шлют логи напрямую в хранилище, например в Loki или Elasticsearch.

- Менее сложная схема для использования.
- Доступна из коробки без лишних зависимостей, кроме хранилища.
- Сложные трансформации потребляют больше ресурсов на узлах для приложений.

Централизованная

Все логи отсылаются в один из доступных агрегаторов, например, Logstash, Vector. Агенты на узлах стараются отправить логи с узла максимально быстро с минимальным потреблением ресурсов. Сложные преобразования применяются на стороне агрегатора.

- Меньшее потребление ресурсов для приложений на узлах.
- Пользователи могут настроить в агрегаторе любые трансформации и слать логи в гораздо большее количество хранилищ.
- Количество выделенных узлов под агрегаторы может увеличиваться или уменьшаться в зависимости от нагрузки.

Потоковая

Главная задача этой архитектуры — как можно быстрее отправить логи в очередь сообщений, из которой они в служебном порядке будут переданы в долгосрочное хранилище для дальнейшего анализа.

- Те же плюсы и минусы, что и у централизованной архитектуры, но добавляется еще одно промежуточное хранилище.
- Повышенная надежность. Подходит тем, для кого доставка логов является наиболее критичной.

Метаданные

При сборе логов сообщения будут обогащены метаданными в зависимости от способа их сбора. Обогащение происходит на этапе `Source`.

Kubernetes

Следующие поля будут экспортированы:

Label	Pod spec path
pod	metadata.name
namespace	metadata.namespace
pod_labels	metadata.labels
pod_ip	status.podIP
image	spec.containers[].image
container	spec.containers[].name
node	spec.nodeName
pod_owner	metadata.ownerRef[0]
Label	Node spec path
node_group	metadata.labels[].node.deckhouse.io/group

i Для Splunk поля `pod_labels` не экспортируются, потому что это вложенный объект, который не поддерживается в Splunk.

File

Следующие поля будут экспортированы:

Label	Pod spec path
host	spec.nodeName
host_ip	status.hostIP
file	путь к читаемому файлу логов

Фильтры сообщений

Существуют два фильтра для снижения количества отправляемых сообщений в хранилище, — `log filter` и `label filter`.

Они запускаются сразу после объединения строк с помощью multiline parser.

- `label filter` — правила запускаются для метаданных сообщения. Поля для метаданных (или лейблов) наполняются на основании источника логов, и для разных источников будет разный набор полей. Эти правила необходимы, например, чтобы отбросить сообщения от определенного контейнера или пода с/без какого-то лейбла.
- `log filter` — правила запускаются для исходного сообщения. Существует возможность отбрасывать сообщение на основе поля JSON или, если сообщение не имеет формата JSON, использовать регулярное выражение для поиска в строке.

Оба фильтра имеют одинаковую структурированную конфигурацию:

- `field` — источник данных для запуска фильтрации (чаще всего это значение лейбла или параметра из JSON-документа).
- `operator` — действие для сравнения, доступные варианты — `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`.
- `values` — определяет разные значения для разных операторов:
 - `DoesNotExist`, `Exists` — не поддерживаются;
 - `In`, `NotIn` — значение поля должно равняться или не равняться одному из значений в списке `values`;
 - `Regex`, `NotRegex` — значение должно подходить хотя бы под одно или не подходить ни под одно регулярное выражение из списка `values`.

Найти больше примеров можно в разделе Примеры документации.



Extra labels добавляются на этапе `Destination`, поэтому невозможно фильтровать логи на их основании.

Настройки

Модуль начинает чтение логов, только если создан pipeline в виде связанных между собой `ClusterLoggingConfig/PodLoggingConfig` и `ClusterLogDestination`.

В модуле реализовано 6 алертов.

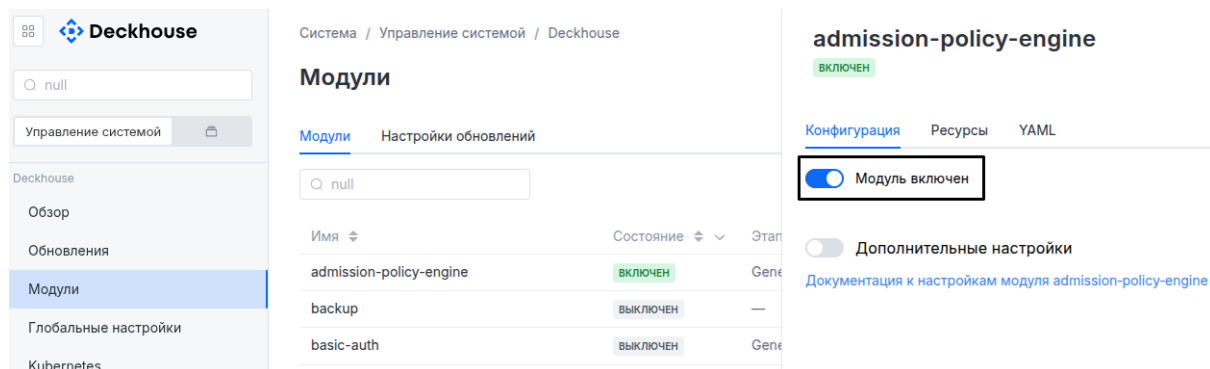
Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `log-shipper`, включите (или выключите) переключатель «Модуль включен».

Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `log-shipper`:

```
d8 system module enable log-shipper
```

- **С помощью ModuleConfig log-shipper.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `log-shipper` (создайте его, при необходимости).

Пример манифеста для включения модуля `log-shipper`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  enabled: true
```

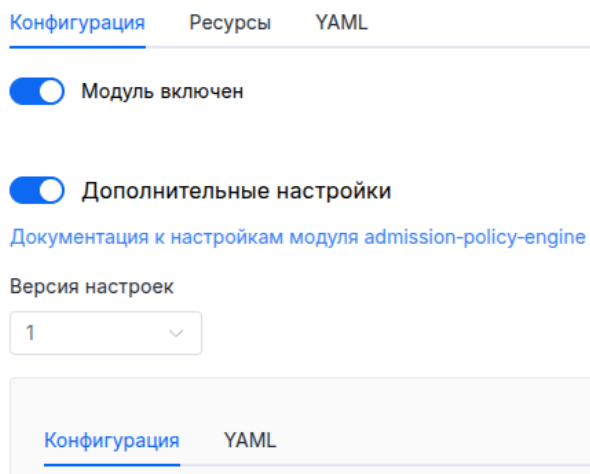
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `log-shipper`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `log-shipper` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `log-shipper`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `log-shipper` (подробнее о настройке Deckhouse читайте

в документации), выполнив следующую команду:

```
d8 k edit mc log-shipper
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `log-shipper`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `log-shipper`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Перенесите `.settings.resourcesRequests.static.cpu` и `.settings.resourcesRequests.static.memory` в `.settings.resourcesRequests.static.requests`.

Параметры

Версия схемы: 2

- **settings**

объект

- **settings.debug**

булевый

Включить или выключить отладочные логи агента. По умолчанию: `false`.

По умолчанию: `false`

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.resourcesRequests**

объект

Настройки максимальных значений CPU и памяти, которые может запросить под при выборе узла (CPU requests, memory requests).

Если модуль `vertical-pod-autoscaler` выключен, максимальные значения становятся значениями по умолчанию.

Примеры:

```
resourcesRequests:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
    memory:
      min: 256Mi
      max: 2Gi
```

```
resourcesRequests:
  mode: VPA
  vpa:
    mode: Initial
    cpu:
      min: 1
      max: 3000m
    memory:
      min: 1024
      max: 4096
```

```
resourcesRequests:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

■ **settings.resourcesRequests.mode**

строка

Обязательный параметр

Режим управления запросами ресурсов (requests):

- **Static** — классический, с помощью явного указания ограничения. Настраивается в секции параметров `static`;
- **VPA** — с помощью VPA. Настраивается в секции параметров `vpa`.

По умолчанию: `VPA`

Допустимые значения: `VPA`, `Static`

■ **settings.resourcesRequests.static**

объект

Настройка управления ресурсами в режиме `Static`.

■ **settings.resourcesRequests.static.limits**

объект

Настройка ограничений CPU и памяти (limits).

■ **settings.resourcesRequests.static.limits.cpu**

строка от число

Настройка ограничения CPU (CPU limits).

Шаблон: `^[0-9]+m?$`

■ **settings.resourcesRequests.static.limits.memory**

строка от число

Настройка ограничения памяти (memory limits).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **settings.resourcesRequests.static.requests**

объект

Настройка запросов ресурсов (requests) для подов.

■ **settings.resourcesRequests.static.requests.cpu**

строка от число

Настройка запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

■ **settings.resourcesRequests.static.requests.memory**

строка от число

Настройка запроса памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **settings.resourcesRequests.vpa**

объект

Настройка управления ресурсами в режиме `VPA`.

■ **settings.resourcesRequests.vpa.cpu**

объект

Обязательный параметр

Настройки VPA при работе с CPU.

- **settings.resourcesRequests.vpa.cpu.max**

строка от число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: 500m

Шаблон: `^[0-9]+m?$`

- **settings.resourcesRequests.vpa.cpu.min**

строка от число

Минимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: 50m

Шаблон: `^[0-9]+m?$`

- **settings.resourcesRequests.vpa.memory**

объект

Обязательный параметр

Настройки VPA при работе с памятью.

- **settings.resourcesRequests.vpa.memory.max**

строка от число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: 2048Mi

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesRequests.vpa.memory.min**

строка от число

Минимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: 64Mi

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesRequests.vpa.mode**

строка

Обязательный параметр

Режим работы VPA.

По умолчанию: Initial

Допустимые значения: Initial, Auto

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**

строка

- **settings.tolerations.operator**

строка

- **settings.tolerations.tolerationSeconds**

целочисленный

- **settings.tolerations.value**

строка

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

ClusterLogDestination

Scope: Cluster

- v1alpha2
- v1alpha1

Describes setting for a log storage, which you can use in many log sources.

`metadata.name` — is an upstream name, which you should use in custom resource `ClusterLoggingConfig`.

- **spec**
объект

Обязательный параметр

- **spec.buffer**
объект

Buffer parameters.

- **spec.buffer.disk**
объект

Disk buffer parameters.

- **spec.buffer.disk.maxSize**
строка или число

The maximum size of the buffer on disk. Must be at least ~256MB (268435488 bytes).

You can express size as a plain integer or as a fixed-point number using one of these quantity suffixes: `E`, `P`, `T`, `G`, `M`, `k`, `Ei`, `Pi`, `Ti`, `Gi`, `Mi`, `Ki`.

More about resource quantity:

- `kubernetes quantity`
- `memory resource units`

Шаблон: `^(\+|-)?((\[0-9]+\([0-9]*\)?|(\.[0-9]+))((\[KMGTPE]i)|[numkMGTPe]|([eE](\+|-)?((\[0-9]+\([0-9]*\)?|(\.[0-9]+))))))?$`

Примеры:

```
maxSize: 512Mi
```

```
maxSize: 268435488
```

- **spec.buffer.memory**
объект

- **spec.buffer.memory.maxEvents**
число

The maximum number of events allowed in the buffer.

- **spec.buffer.type**
строка

Обязательный параметр

The type of buffer to use.

Допустимые значения: `Disk`, `Memory`

- **spec.buffer.whenFull**
строка

Event handling behavior when a buffer is full.

По умолчанию: `Block`

Допустимые значения: `DropNewest`, `Block`

- **spec.elasticsearch**
объект

- **spec.elasticsearch.auth**
объект

- **spec.elasticsearch.auth.awsAccessKey**
строка

Base64-encoded AWS `ACCESS_KEY`.

- **spec.elasticsearch.auth.awsAssumeRole**

строка

The ARN of an IAM role to assume at startup.

- **spec.elasticsearch.auth.awsRegion**

строка

AWS region for authentication.

- **spec.elasticsearch.auth.awsSecretKey**

строка

Base64-encoded AWS `SECRET_KEY`.

- **spec.elasticsearch.auth.password**

строка

Base64-encoded Basic authentication password.

- **spec.elasticsearch.auth.strategy**

строка

The authentication strategy to use.

По умолчанию: `Basic`

Допустимые значения: `Basic`, `AWS`

- **spec.elasticsearch.auth.user**

строка

The Basic authentication user name.

- **spec.elasticsearch.dataStreamEnabled**

булевый

Use for storage indexes or datastreams

(<https://www.elastic.co/guide/en/elasticsearch/reference/master/data-streams.html>).

Datastream usage is better for logs and metrics storage but they works only for Elasticsearch $\geq 7.16.X$.

По умолчанию: `false`

- **spec.elasticsearch.docType**

строка

The `doc_type` for your index data. This is only relevant for Elasticsearch $\leq 6.X$.

- For Elasticsearch $\geq 7.X$ you do not need this option since this version has removed `doc_type` mapping;
- For Elasticsearch $\geq 6.X$ the recommended value is `_doc`, because using it will make it easy to upgrade to 7.X;
- For Elasticsearch $< 6.X$ you can't use a value starting with `_` or empty string. Use, for example, values like `logs`.

- **spec.elasticsearch.endpoint**

строка

Обязательный параметр

Base URL of the Elasticsearch instance.

- **spec.elasticsearch.index**

строка

Index name to write events to.

- **spec.elasticsearch.pipeline**

строка

Name of the pipeline to apply.

- **spec.elasticsearch.tls**

объект

Configures the TLS options for outgoing connections.

- **spec.elasticsearch.tls.caFile**

строка

Base64-encoded CA certificate in PEM format.

- **spec.elasticsearch.tls.clientCrt**

объект

Configures the client certificate for outgoing connections.

- **spec.elasticsearch.tls.clientCrt.crtFile**

строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.elasticsearch.tls.clientCert.keyFile**

строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.elasticsearch.tls.clientCert.keyPass**

строка

Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.elasticsearch.tls.secretRef**

объект

Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

- **spec.elasticsearch.tls.secretRef.name**

строка

Name of the Secret with TLS certificates.

- **spec.elasticsearch.tls.verifyCertificate**

булевый

Validate the TLS certificate of the remote host. Specifically the issuer is checked but not CRLs (Certificate Revocation Lists).

По умолчанию: `true`

- **spec.elasticsearch.tls.verifyHostname**

булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

- **spec.extraLabels**

объект

A set of labels that will be attached to each batch of events.

You can use simple templating here: `{{ app }}`.

There are some reserved keys:

- `parsed_data`
- `pod`
- `pod_labels_*`
- `pod_ip`
- `namespace`
- `image`
- `container`
- `node`
- `pod_owner`

More about field path notation...

Пример:

```
extraLabels:
  forwarder: vector
  key: value
  app_info: "{{ app }}"
  array_member: "{{ array[0] }}"
  symbol_escating_value: "{{ pay\\.day }}"
```

- **spec.kafka**

объект

- **spec.kafka.bootstrapServers**

массив строк

Обязательный параметр

A list of host and port pairs that are the addresses of the Kafka brokers in a “bootstrap” Kafka cluster that a Kafka client connects to initially to bootstrap itself.

По умолчанию: `[]`

Пример:

```
bootstrapServers:
- 10.14.22.123:9092
- 10.14.23.332:9092
```

- **Элемент массива**

строка

Шаблон: `^(.+)\:\d{1,5}$`

- **spec.kafka.encoding**

объект

How to encode the message.

- **spec.kafka.encoding.cef**

объект

CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

- **spec.kafka.encoding.cef.deviceProduct**

строка

По умолчанию: `log-shipper-agent`

- **spec.kafka.encoding.cef.deviceVendor**

строка

По умолчанию: `Deckhouse`

- **spec.kafka.encoding.cef.deviceVersion**

строка

По умолчанию: `1`

- **spec.kafka.encoding.codec**

строка

Available encoding formats.

По умолчанию: `JSON`Допустимые значения: `JSON`, `CEF`

- **spec.kafka.keyField**

строка

Allows to set the `key_field`.

Примеры:

```
keyField: host
```

```
keyField: node
```

```
keyField: namespace
```

```
keyField: parsed_data.app_info
```

- **spec.kafka.sasl**

объект

Configuration for SASL authentication when interacting with Kafka.

- **spec.kafka.sasl.mechanism**

строка

Обязательный параметр

The SASL mechanism to use. Only PLAIN and SCRAM-based mechanisms are supported.

Допустимые значения: `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`

- **spec.kafka.sasl.password**

строка

Обязательный параметр

The SASL password.

Пример:

```
password: qwerty
```

- **spec.kafka.sasl.username**

строка

Обязательный параметр

The SASL username.

Пример:

```
username: username
```

■ **spec.kafka.tls**

объект

Configures the TLS options for outgoing connections.

■ **spec.kafka.tls.caFile**

строка

Base64-encoded CA certificate in PEM format.

■ **spec.kafka.tls.clientCert**

объект

Configures the client certificate for outgoing connections.

■ **spec.kafka.tls.clientCert.crtFile**

строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

■ **spec.kafka.tls.clientCert.keyFile**

строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

■ **spec.kafka.tls.clientCert.keyPass**

строка

Base64-encoded pass phrase used to unlock the encrypted key file.

■ **spec.kafka.tls.secretRef**

объект

Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

■ **spec.kafka.tls.secretRef.name**

строка

Name of the Secret with TLS certificates.

■ **spec.kafka.tls.verifyCertificate**

булевый

Validate the TLS certificate of the remote host.

По умолчанию: `true`

■ **spec.kafka.tls.verifyHostname**

булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

■ **spec.kafka.topic**

строка

Обязательный параметр

The Kafka topic name to write events to. This parameter supports template syntax, which enables you to use dynamic per-event values.

Примеры:

```
topic: logs
```

```
topic: logs-{{unit}}-%Y-%m-%d
```

○ **spec.logstash**

объект

■ **spec.logstash.endpoint**

строка

Обязательный параметр

Base URL of the Logstash instance.

■ **spec.logstash.tls**
объект

Configures the TLS options for outgoing connections.

■ **spec.logstash.tls.caFile**
строка

Base64-encoded CA certificate in PEM format.

■ **spec.logstash.tls.clientCert**
объект

Configures the client certificate for outgoing connections.

■ **spec.logstash.tls.clientCert.crtFile**
строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

■ **spec.logstash.tls.clientCert.keyFile**
строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

■ **spec.logstash.tls.clientCert.keyPass**
строка

Base64-encoded pass phrase used to unlock the encrypted key file.

■ **spec.logstash.tls.secretRef**
объект

Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

■ **spec.logstash.tls.secretRef.name**
строка

Name of the Secret with TLS certificates.

■ **spec.logstash.tls.verifyCertificate**
булевый

Validate the TLS certificate of the remote host.

По умолчанию: `true`

■ **spec.logstash.tls.verifyHostname**
булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

○ **spec.loki**
объект

■ **spec.loki.auth**
объект

■ **spec.loki.auth.password**
строка

Base64-encoded Basic authentication password.

■ **spec.loki.auth.strategy**
строка

The authentication strategy to use.

По умолчанию: `Basic`

Допустимые значения: `Basic`, `Bearer`

■ **spec.loki.auth.token**
строка

The token to use for Bearer authentication.

■ **spec.loki.auth.user**

строка

The Basic authentication user name.

- **spec.loki.endpoint**

строка

Обязательный параметр

Base URL of the Loki instance.

i Agent automatically adds `/loki/api/v1/push` into URL during data transmission.

- **spec.loki.tenantID**

строка

ID of a tenant.

This option is used only for GrafanaCloud. When running Loki locally, a tenant ID is not required.

- **spec.loki.tls**

объект

Configures the TLS options for outgoing connections.

- **spec.loki.tls.caFile**

строка

Base64-encoded CA certificate in PEM format.

- **spec.loki.tls.clientCrt**

объект

Configures the client certificate for outgoing connections.

- **spec.loki.tls.clientCrt.crtFile**

строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.loki.tls.clientCrt.keyFile**

строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.loki.tls.clientCrt.keyPass**

строка

Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.loki.tls.secretRef**

объект

Reference to a Kubernetes Secret containing the CA certificate (`ca.pem`), client certificate (`crt.pem`), private key (`key.pem`) and key pass (`keyPass`) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

- **spec.loki.tls.secretRef.name**

строка

Name of the Secret with TLS certificates.

- **spec.loki.tls.verifyCertificate**

булевый

Validate the TLS certificate of the remote host.

If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

По умолчанию: `true`

- **spec.loki.tls.verifyHostname**

булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

- **spec.rateLimit**

объект

Parameter for limiting the flow of events.

- **spec.rateLimit.excludes**

массив объектов

List of excludes for keyField.

Only NOT matched log entries would be rate limited.

Примеры:

```
excludes:
  field: tier
  operator: Exists
```

```
excludes:
  field: foo
  operator: NotIn
  values:
    - dev
    - 42
    - 'true'
    - '3.14'
```

```
excludes:
  field: bar
  operator: Regex
  values:
    - "^abc"
    - "^\\d.+ $"
```

- **spec.rateLimit.excludes.field**

строка

Обязательный параметр

Field name for filtering.

- **spec.rateLimit.excludes.operator**

строка

Обязательный параметр

Operator for log field comparisons:

- **In** — finds a substring in a string.
- **NotIn** — is a negative version of the **In** operator.
- **Regex** — is trying to match regexp over the field; only log events with matching fields will pass.
- **NotRegex** — is a negative version of the **Regex** operator; log events without fields or with not matched fields will pass.
- **Exists** — drops log event if it contains some fields.
- **DoesNotExist** — drops log event if it does not contain some fields.

Допустимые значения: **In**, **NotIn**, **Regex**, **NotRegex**, **Exists**, **DoesNotExist**

- **spec.rateLimit.excludes.values**

массив

Array of values or regexes for corresponding operations. Does not work for **Exists** and **DoesNotExist** operations.

Fields with float or boolean values will be converted to strings during comparison.

- **spec.rateLimit.keyField**

строка

The name of the log field whose value will be hashed to determine if the event should be rate limited.

- **spec.rateLimit.linesPerMinute**

число

Обязательный параметр

The number of records per minute.

- **spec.socket**

объект

- **spec.socket.address**

строка

Обязательный параметр

Address of the socket.

Шаблон: `^.*:[1-9][0-9]+$`

- **spec.socket.encoding**

объект

How to encode the message.

- **spec.socket.encoding.cef**

объект

CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

- **spec.socket.encoding.cef.deviceProduct**

строка

По умолчанию: `log-shipper-agent`

- **spec.socket.encoding.cef.deviceVendor**

строка

По умолчанию: `Deckhouse`

- **spec.socket.encoding.cef.deviceVersion**

строка

По умолчанию: `1`

- **spec.socket.encoding.codec**

строка

Available encoding formats.

По умолчанию: `JSON`

Допустимые значения: `Text`, `JSON`, `Syslog`, `CEF`, `GELF`

- **spec.socket.mode**

строка

Обязательный параметр

Допустимые значения: `TCP`, `UDP`

- **spec.socket.tcp**

объект

- **spec.socket.tcp.tls**

объект

Configures the TLS options for outgoing connections.

- **spec.socket.tcp.tls.caFile**

строка

Base64-encoded CA certificate in PEM format.

- **spec.socket.tcp.tls.clientCert**

объект

Configures the client certificate for outgoing connections.

- **spec.socket.tcp.tls.clientCert.crtFile**

строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.socket.tcp.tls.clientCert.keyFile**

строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.socket.tcp.tls.clientCert.keyPass**

строка

Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.socket.tcp.tls.secretRef**

объект

Reference to a Kubernetes Secret containing the CA certificate (`ca.pem`), client certificate (`crt.pem`), private key (`key.pem`) and key pass (`keyPass`) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

- **spec.socket.tcp.tls.secretRef.name**

строка

Name of the Secret with TLS certificates.

- **spec.socket.tcp.verifyCertificate**

булевый

Validate the TLS certificate of the remote host.

If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

По умолчанию: `true`

- **spec.socket.tcp.verifyHostname**
булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

- **spec.splunk**

объект

- **spec.splunk.endpoint**
строка

Обязательный параметр

Base URL of the Splunk instance.

Пример:

```
endpoint: https://http-inputs-hec.splunkcloud.com
```

- **spec.splunk.index**
строка

Index name to write events to.

- **spec.splunk.tls**
объект

Configures the TLS options for outgoing connections.

- **spec.splunk.tls.caFile**
строка

Base64-encoded CA certificate in PEM format.

- **spec.splunk.tls.clientCert**
объект

Configures the client certificate for outgoing connections.

- **spec.splunk.tls.clientCert.crtFile**
строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.splunk.tls.clientCert.keyFile**
строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.splunk.tls.clientCert.keyPass**
строка

Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.splunk.tls.secretRef**
объект

Reference to a Kubernetes Secret containing the CA certificate (`ca.pem`), client certificate (`crt.pem`), private key (`key.pem`) and key pass (`keyPass`) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

- **spec.splunk.tls.secretRef.name**
строка

Name of the Secret with TLS certificates.

- **spec.splunk.tls.verifyCertificate**
булевый

Validate the TLS certificate of the remote host.

По умолчанию: `true`

- **spec.splunk.tls.verifyHostname**
булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

■ **spec.splunk.token**

строка

Обязательный параметр

Default Splunk HEC token. If an event has a token set in its metadata, it will have priority over the one set here.

○ **spec.transformations**

массив объектов

Ordered list of transformations applied to each log event before it is sent to the destination. Items are executed in array order.

`AddLabels` runs after `extraLabels` when both are configured.

Available actions:

- `ReplaceKeys` — recursively replace matches of the `source` pattern with `target` in the given paths.
- `ParseMessage` — parse the `message` field using `sourceFormat` and write the result to `targetLabel` (default `.message`; the lone `.` merges into the event root). On parse failure the original `message` is left unchanged.
- `DropLabels` — remove the value at each `label` path, or if `keepKeys` is set on an item, keep only those child key names inside the object at `label`.
- `AddLabels` — add labels (literals, `{{ .path }}` templates, optional `when`). Runs after `extraLabels`; on key name conflicts `AddLabels` wins.
- `ReplaceValue` — replace substrings in scalar string fields using the `source` regex; `target` may reference named groups as `{{ group_name }}`.

Примеры:

```
transformations:
  action: ReplaceKeys
  replaceKeys:
    source: "."
    target: _
    labels:
      - ".pod_labels"
```

```
transformations:
  action: ParseMessage
  parseMessage:
    sourceFormat: String
    string:
      regex: "^(?P<msg>.*)$"
    setLabels:
      msg: "{{ msg }}"
```

```
transformations:
  action: ParseMessage
  parseMessage:
    sourceFormat: Klog
```

```
transformations:
  action: ParseMessage
  parseMessage:
    sourceFormat: JSON
    json:
      depth: 1
```

```
transformations:
  action: DropLabels
  dropLabels:
    labels:
      - label: ".first"
      - label: ".second"
      - label: ".pod_labels"
    keepKeys:
      - app
      - team
```

```
transformations:
  action: AddLabels
  addLabels:
    when:
      - .namespace == "production"
  setLabels:
    ".env": prod
    ".source_app": "{{ .pod_labels.app }}"
```

```
transformations:
  action: ReplaceValue
  replaceValue:
    source: "\\d+"
    target: REDACTED
  labels:
    - ".message"
```

■ **spec.transformations.action**

строка

Transformation type; selects which parameter block is used.

Допустимые значения: `ReplaceKeys`, `ParseMessage`, `DropLabels`, `AddLabels`, `ReplaceValue`

■ **spec.transformations.addLabels**

объект

Adds fields to the event from `setLabels` (literals and `{{ .path }}` templates). Runs after `extraLabels` when both are set; on path conflicts values from this transform win. Optional `when` filter.

■ **spec.transformations.addLabels.setLabels**

объект

Обязательный параметр

Keys are destination label paths in Vector notation: each key starts with `.` relative to the event root (e.g. `.env`, `.source_app`). Values are literals or `{{ .path }}` templates where `path` in the template is a source field path with a leading `.`.

Примеры:

```
setLabels:
  ".tier": backend
```

```
setLabels:
  ".source_app": "{{ .pod_labels.app }}"
```

■ **spec.transformations.addLabels.when**

массив строк

Optional list of conditions combined with logical AND. In comparisons, the field path is a leading dot and dot-separated segments (e.g. `.namespace`, `.pod_labels.app`). Each item is either: a comparison `path ==|!=|~=|!~= value` (regular expression for `~=` / `!~=`), a `path` with no operator or right-hand side (field exists), or `!.path` (field does not exist).

Примеры:

```
when: .namespace == "production"
```

```
when: .pod_labels.app =~ "^api-"
```

```
when: .level != "debug"
```

```
when: ".pod_labels.team"
```

```
when: "!.pod_labels.legacy"
```

■ **Элемент массива**

строка

Шаблон: `^[a-zA-Z0-9_\s\\[\]\(\)\{\}\?*\+.\^\$\\-!:\,\'\"|<>~#/@]+$`

■ **spec.transformations.dropLabels**

объект

Removes field paths, or narrows objects by keeping only selected child keys at each `label` when `keepKeys` is set.

■ **spec.transformations.dropLabels.labels**

массив объектов

Обязательный параметр

Each item has a required Vector path `label` and optional `keepKeys`. If `keepKeys` is omitted, the subtree at `label` is removed. If `keepKeys` is set, only those child key names are kept in the object at `label`.

- **spec.transformations.dropLabels.labels.keepKeys**

массив строк

Child key names (no leading dot) to keep inside the object at `label`; all other child keys are removed.

- **Элемент массива**

строка

Шаблон: `^[a-zA-Z0-9_.\-/\]+$`

- **spec.transformations.dropLabels.labels.label**

строка

Обязательный параметр

Vector path to a field or subtree; when `keepKeys` is set, must resolve to an object.

Шаблон: `^\.[a-zA-Z0-9_\\[\]\-\-"/:]+\.`

- **spec.transformations.parseMessage**

объект

Parses the string in the `message` field (format from `sourceFormat`) and writes the result to `targetLabel` as a Vector path with a leading `.` (default `.message`; the lone `.` merges the parsed object into the event root). On parse failure the original `message` field is left unchanged.

- **spec.transformations.parseMessage.json**

объект

- **spec.transformations.parseMessage.json.depth**

целочисленный

Depth for JSON parsing.

Допустимые значения: `1 <= X <= 128`

- **spec.transformations.parseMessage.sourceFormat**

строка

Обязательный параметр

Message format for converting into an object.

Допустимые значения: `String`, `JSON`, `Klog`, `SysLog`, `CLF`, `Logfmt`

- **spec.transformations.parseMessage.string**

объект

Settings for `sourceFormat`: `String`. `regex` and `setLabels` are required.

- **spec.transformations.parseMessage.string.regex**

строка

Обязательный параметр

Regular expression with named capture groups; requires `setLabels`.

Шаблон: `^[a-zA-Z0-9_\\s\\[\\](){}?+.*$\\-!=:;'\\"|<>~#/@]+$`

- **spec.transformations.parseMessage.string.setLabels**

объект

Обязательный параметр

Maps output keys to literals or `{{ group_name }}` templates.

- **spec.transformations.parseMessage.targetLabel**

строка

Destination path for the parsed value (default `.message`). The lone value `.` merges into the event root. Must not be `.parsed_data`.

По умолчанию: `.message`

Шаблон: `^(\\.|\\.[a-zA-Z0-9_\\[\\]\\-\\-"/:']+$`

Примеры:

```
targetLabel: ".message"
```

```
targetLabel: "."
```

```
targetLabel: ".tmp"
```

- **spec.transformations.replaceKeys**

объект

Recursively replaces matches of the `source` pattern with the `target` value in the listed paths (`labels`). Paths use Vector notation with a leading `.` on each list item.

- **spec.transformations.replaceKeys.labels**

массив строк

Обязательный параметр

Label paths where the replacement runs. Each item must start with `.` (Vector field path notation, e.g. `.pod_labels`, `.parsed_data`).

Примеры:

```
labels: ".pod_labels"
```

```
labels: ".annotations"
```

- **Элемент массива**

строка

Шаблон: `^\.[a-zA-Z0-9_\[\]\.\-"/:']+$`

- **spec.transformations.replaceKeys.source**

строка

Обязательный параметр

Pattern used to find matches. Can be a static string or a regular expression (same character set as other regex fields in transformations).

Шаблон: `^[a-zA-Z0-9_\s\[\](){}?*.^$\-=:;', '<>~#/@]+$`

Примеры:

```
source: "."
```

```
source: "^old\\."
```

- **spec.transformations.replaceKeys.target**

строка

Literal replacement string for matched substrings (not a regex).

По умолчанию:

Примеры:

```
target: _
```

```
target: ''
```

- **spec.transformations.replaceValue**

объект

Replace matches of `source` in each listed scalar string field with `target` .

- **spec.transformations.replaceValue.labels**

массив строк

Обязательный параметр

Paths to scalar string fields where replacement runs. Each item must start with `.` (Vector path, e.g. `.message`, `.parsed_data.token`).

Примеры:

```
labels: ".message"
```

```
labels: ".parsed_data.line"
```

- **Элемент массива**

строка

Шаблон: `^\.[a-zA-Z0-9_\[\]\.\-"/:']+$`

- **spec.transformations.replaceValue.source**

строка

Обязательный параметр

Regular expression to search in each scalar field value.

Шаблон: `^[a-zA-Z0-9_\s\[\](){}?*.^$\-=:;', '<>~#/@]+$`

Примеры:

```
source: "\\d+"
```

```
source: "(?P<tok>token=[^\\s]+) "
```

- **spec.transformations.replaceValue.target**
строка

Обязательный параметр

Replacement string; may use `{{name}}` for named capture groups.

Примеры:

```
target: REDACTED
```

```
target: "{{tok}}"
```

- **spec.type**
строка

Type of a log storage backend.

Допустимые значения: `Loki`, `Elasticsearch`, `Logstash`, `Vector`, `Kafka`, `Splunk`, `Socket`

- **spec.vector**
объект

- **spec.vector.endpoint**
строка

Обязательный параметр

An address of the Vector instance. API v2 must be used for communication between instances.

Шаблон: `^(.+):([0-9]{1,5})$`

- **spec.vector.tls**
объект

Configures the TLS options for outgoing connections.

- **spec.vector.tls.caFile**
строка

Base64-encoded CA certificate in PEM format.

- **spec.vector.tls.clientCert**
объект

Configures the client certificate for outgoing connections.

- **spec.vector.tls.clientCert.crtFile**
строка

Обязательный параметр

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.vector.tls.clientCert.keyFile**
строка

Обязательный параметр

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.vector.tls.clientCert.keyPass**
строка

Base64-encoded passphrase used to unlock the encrypted key file.

- **spec.vector.tls.secretRef**
объект

Reference to a Kubernetes Secret containing the CA certificate (`ca.pem`), client certificate (`crt.pem`), private key (`key.pem`) and key pass (`keyPass`) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in `d8-log-shipper` namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

- **spec.vector.tls.secretRef.name**
строка

Name of the Secret with TLS certificates.

- **spec.vector.tls.verifyCertificate**
булевый

Validate the TLS certificate of the remote host.

По умолчанию: `true`

- **spec.vector.tls.verifyHostname**
булевый

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

По умолчанию: `true`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описывает параметры хранилища логов, к которому можно подключить бесконечное количество источников.

`metadata.name` задаёт имя upstream'a, который будет использоваться в ресурсе `ClusterLoggingConfig`.

- **spec**
объект

Обязательный параметр

- **spec.buffer**
объект

Параметры буфера.

- **spec.buffer.disk**
объект

Параметры буфера на диске.

- **spec.buffer.disk.maxSize**
строка или число

Максимальный размер буфера на диске. Должен быть не меньше ~256 МБ (268435488 байт).

Вы можете указать размер как простое целое число или как число с одним из следующих суффиксов: E, P, T, G, M, k, Ei, Pi, Ti, Gi, Mi, Ki.

Подробнее об указании ресурсов:

- количество kubernetes;
- единицы ресурсов памяти.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+)(([KMGTPE]i)|[numkMGTPe]|([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))))?$`

Примеры:

```
maxSize: 512Mi
```

```
maxSize: 268435488
```

- **spec.buffer.memory**
объект
 - **spec.buffer.memory.maxEvents**
число

Максимальное количество событий в буфере.

- **spec.buffer.type**
строка

Обязательный параметр

Тип буфера для использования.

Допустимые значения: `Disk`, `Memory`

- **spec.buffer.whenFull**
строка

Поведение при заполнении буфера.

По умолчанию: `Block`

Допустимые значения: `DropNewest`, `Block`

- **spec.elasticsearch**
объект

- **spec.elasticsearch.auth**
объект
 - **spec.elasticsearch.auth.awsAccessKey**
строка
Закодированный в Base64 AWS `ACCESS_KEY`.
 - **spec.elasticsearch.auth.awsAssumeRole**
строка

ARN роли в IAM.

- **spec.elasticsearch.auth.awsRegion**
строка

Регион AWS для аутентификации.

- **spec.elasticsearch.auth.awsSecretKey**
строка

Закодированный в Base64 AWS `SECRET_KEY`.

- **spec.elasticsearch.auth.password**
строка

Закодированный в Base64 пароль для Basic-аутентификации.

- **spec.elasticsearch.auth.strategy**
строка

Тип аутентификации — `Basic` или `AWS`.

По умолчанию: `Basic`

Допустимые значения: `Basic`, `AWS`

- **spec.elasticsearch.auth.user**
строка

Имя пользователя, используемое при Basic-аутентификации.

- **spec.elasticsearch.dataStreamEnabled**
булевый

Использовать Datastream для хранения логов
(<https://www.elastic.co/guide/en/elasticsearch/reference/master/data-streams.html>).

Datastreams производительнее для хранения логов и метрик, но они существуют только в Elasticsearch $\geq 7.16.X$.

По умолчанию: `false`

- **spec.elasticsearch.docType**
строка

Использовать `doc_type` для индексов. Имеет смысл использовать только для Elasticsearch $\leq 6.X$.

- Для Elasticsearch $\geq 7.X$ вам не нужно использовать эту опцию, так как все работает по умолчанию.
- Для Elasticsearch $\geq 6.X$ рекомендуемое значение — `_doc`, так как его использование позволит легко обновиться до Elasticsearch версии 7.X.
- Для Elasticsearch $< 6.X$ вы можете использовать любое значение, которое не начинается с `_`. Например, `logs`.

- **spec.elasticsearch.endpoint**
строка

Обязательный параметр

URL для подключения к Elasticsearch.

- **spec.elasticsearch.index**
строка

Имя индекса, куда будут сохранены данные. Это поле можно задать динамически.

- **spec.elasticsearch.pipeline**
строка

Имя pipeline, который будет применен к данным перед записью в индекс.

- **spec.elasticsearch.tls**
объект

Настройки защищенного TLS-соединения.

- **spec.elasticsearch.tls.caFile**
строка

Закодированный в Base64 сертификат CA в формате PEM.

- **spec.elasticsearch.tls.clientCrt**
объект

Конфигурация клиентского сертификата.

- **spec.elasticsearch.tls.clientCrt.crtFile**
строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

- **spec.elasticsearch.tls.clientCert.keyFile**

строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

- **spec.elasticsearch.tls.clientCert.keyPass**

строка

Закодированный в Base64 пароль для ключа.

- **spec.elasticsearch.tls.secretRef**

объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (ca.pem), клиентский сертификат (crt.pem), приватный ключ (key.pem) и пароль ключа для клиентского сертификата (keyPass) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.elasticsearch.tls.secretRef.name**

строка

Имя секрета с TLS сертификатами.

- **spec.elasticsearch.tls.verifyCertificate**

булевый

Проверка TLS-сертификата удаленного хоста. Сертификат не будет проверен на наличие в списках отозванных сертификатов (Certificate Revocation Lists).

По умолчанию: `true`

- **spec.elasticsearch.tls.verifyHostname**

булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

- **spec.extraLabels**

объект

Дополнительные ярлыки, которыми будут снабжаться записи логов.

Вы можете использовать простые шаблоны: `{{ app }}`.

Некоторые ключи зарезервированы:

- `parsed_data`;
- `pod`;
- `pod_labels_*`;
- `pod_ip`;
- `namespace`;
- `image`;
- `container`;
- `node`;
- `pod_owner`.

Подробнее о путях к полям...

Пример:

```
extraLabels:
  forwarder: vector
  key: value
  app_info: "{{ app }}"
  array_member: "{{ array[0] }}"
  symbol_escating_value: "{{ pay\\.day }}"
```

- **spec.kafka**

объект

- **spec.kafka.bootstrapServers**

массив строк

Обязательный параметр

Список пар адресов (хост:порт) Kafka-брокеров в кластере Kafka, к которым должны подключиться клиенты для получения метаданных (топиков и партиций).

По умолчанию: `[]`

Пример:

```
bootstrapServers:
- 10.14.22.123:9092
- 10.14.23.332:9092
```

- **Элемент массива**
строка

Шаблон: `^(.+)\:\d{1,5}$`

- **spec.kafka.encoding**
объект

В каком формате закодировать сообщение.

- **spec.kafka.encoding.cef**
объект

Поля конфигурации, специфичные для CEF. Применяется только в том случае, если для кодирования установлено значение `CEF`.

- **spec.kafka.encoding.cef.deviceProduct**
строка

По умолчанию: `log-shipper-agent`

- **spec.kafka.encoding.cef.deviceVendor**
строка

По умолчанию: `Deckhouse`

- **spec.kafka.encoding.cef.deviceVersion**
строка

По умолчанию: `1`

- **spec.kafka.encoding.codec**
строка

Доступные форматы для кодирования.

По умолчанию: `JSON`

Допустимые значения: `JSON`, `CEF`

- **spec.kafka.keyField**
строка

Позволяет задать поле `key_field`.

Примеры:

```
keyField: host
```

```
keyField: node
```

```
keyField: namespace
```

```
keyField: parsed_data.app_info
```

- **spec.kafka.sasl**
объект

Конфигурация аутентификации SASL для взаимодействия с Kafka.

- **spec.kafka.sasl.mechanism**
строка

Обязательный параметр

Механизм аутентификации SASL. Поддерживаются только PLAIN и SCRAM-подобные механизмы.

Допустимые значения: `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`

- **spec.kafka.sasl.password**
строка

Обязательный параметр

Пароль пользователя.

Пример:

```
password: qwerty
```

- **spec.kafka.sasl.username**
строка

Обязательный параметр

Имя пользователя.

Пример:

```
username: username
```

■ **spec.kafka.tls**

объект

Настройки защищенного TLS-соединения.

■ **spec.kafka.tls.caFile**

строка

Закодированный в Base64 сертификат CA в формате PEM.

■ **spec.kafka.tls.clientCert**

объект

Конфигурация клиентского сертификата.

■ **spec.kafka.tls.clientCert.crtFile**

строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

■ **spec.kafka.tls.clientCert.keyFile**

строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

■ **spec.kafka.tls.clientCert.keyPass**

строка

Закодированный в Base64 пароль для ключа.

■ **spec.kafka.tls.secretRef**

объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (`ca.pem`), клиентский сертификат (`crt.pem`), приватный ключ (`key.pem`) и пароль ключа для клиентского сертификата (`keyPass`) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

■ **spec.kafka.tls.secretRef.name**

строка

Имя секрета с TLS-сертификатами.

■ **spec.kafka.tls.verifyCertificate**

булевый

Проверка действия TLS-сертификата удаленного хоста.

По умолчанию: `true`

■ **spec.kafka.tls.verifyHostname**

булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

■ **spec.kafka.topic**

строка

Обязательный параметр

Имя топика в Kafka для записи логов. Этот параметр поддерживает синтаксис шаблонов, что дает возможность динамического создания топиков.

Примеры:

```
topic: logs
```

```
topic: logs-{{unit}}-%Y-%m-%d
```

○ **spec.logstash**

объект

■ **spec.logstash.endpoint**

строка

Обязательный параметр

URL для подключения к Logstash.

- **spec.logstash.tls**
объект

Настройки защищенного TLS-соединения.

- **spec.logstash.tls.caFile**
строка

Закодированный в Base64 сертификат CA в формате PEM.

- **spec.logstash.tls.clientCert**
объект

Конфигурация клиентского сертификата.

- **spec.logstash.tls.clientCert.crtFile**
строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

- **spec.logstash.tls.clientCert.keyFile**
строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

- **spec.logstash.tls.clientCert.keyPass**
строка

Закодированный в Base64 пароль для ключа.

- **spec.logstash.tls.secretRef**
объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (`ca.pem`), клиентский сертификат (`crt.pem`), приватный ключ (`key.pem`) и пароль ключа для клиентского сертификата (`keyPass`) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.logstash.tls.secretRef.name**
строка

Имя секрета с TLS-сертификатами.

- **spec.logstash.tls.verifyCertificate**
булевый

Проверка действия TLS-сертификата удаленного хоста.

По умолчанию: `true`

- **spec.logstash.tls.verifyHostname**
булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

- **spec.loki**
объект

- **spec.loki.auth**
объект

- **spec.loki.auth.password**
строка

Закодированный в Base64 пароль для Basic-аутентификации.

- **spec.loki.auth.strategy**
строка

Используемый тип аутентификации.

По умолчанию: `Basic`

Допустимые значения: `Basic`, `Bearer`

- **spec.loki.auth.token**
строка

Токен для Bearer-аутентификации.

- **spec.loki.auth.user**

строка

Имя пользователя, используемое при Basic-аутентификации.

- **spec.loki.endpoint**

строка

Обязательный параметр

URL для подключения к Loki.



Агент автоматически добавляет `/loki/api/v1/push` к URL при отправке данных.

- **spec.loki.tenantID**

строка

ID тенанта.

Эта опция используется только для Grafana Cloud. Для локально запущенного Loki опция ни на что не влияет.

- **spec.loki.tls**

объект

Настройки защищенного TLS-соединения.

- **spec.loki.tls.caFile**

строка

Закодированный в Base64 сертификат CA в формате PEM.

- **spec.loki.tls.clientCrt**

объект

Конфигурация клиентского сертификата

- **spec.loki.tls.clientCrt.crtFile**

строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

- **spec.loki.tls.clientCrt.keyFile**

строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

- **spec.loki.tls.clientCrt.keyPass**

строка

Закодированный в Base64 пароль для ключа.

- **spec.loki.tls.secretRef**

объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (`ca.pem`), клиентский сертификат (`crt.pem`), приватный ключ (`key.pem`) и пароль ключа для клиентского сертификата (`keyPass`) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.loki.tls.secretRef.name**

строка

Имя секрета с TLS-сертификатами.

- **spec.loki.tls.verifyCertificate**

булевый

Проверка TLS-сертификата удаленного хоста.

Если параметр установлен в `false`, сертификат не проверяется на наличие в списке отозванных сертификатов (Certificate Revocation Lists).

По умолчанию: `true`

- **spec.loki.tls.verifyHostname**

булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

- **spec.rateLimit**
объект

Параметр ограничения потока логов, передаваемых в хранилище.

- **spec.rateLimit.excludes**
массив объектов

Список фильтров для keyField.

Только логи, НЕ подпадающие под правила, будут ограничены по количеству записей.

Примеры:

```
excludes:
  field: tier
  operator: Exists
```

```
excludes:
  field: foo
  operator: NotIn
  values:
    - dev
    - 42
    - 'true'
    - '3.14'
```

```
excludes:
  field: bar
  operator: Regex
  values:
    - "^abc"
    - "^\\d.+\\$"
```

- **spec.rateLimit.excludes.field**
строка

Обязательный параметр

Имя поля для фильтрации.

- **spec.rateLimit.excludes.operator**
строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- **In** — ищет строку или элемент в массиве;
- **NotIn** — является инверсией оператора **In**;
- **Regex** — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- **NotRegex** — является инверсией оператора **Regex** (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- **Exists** — проверяет наличие поля и пропускает логи, только если поле есть;
- **DoesNotExist** — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: **In**, **NotIn**, **Regex**, **NotRegex**, **Exists**, **DoesNotExist**

- **spec.rateLimit.excludes.values**
массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций **Exists** и **DoesNotExist**.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

- **spec.rateLimit.keyField**
строка

Имя поля логa, значение которого будет хешировано, чтобы определить, должен ли лог быть ограничен по скорости.

- **spec.rateLimit.linesPerMinute**
число

Обязательный параметр

Количество записей в минуту.

- **spec.socket**
объект

- **spec.socket.address**

строка

Обязательный параметр

Адрес сокета.

Шаблон: `^.*:[1-9][0-9]+$`

■ **spec.socket.encoding**

объект

В каком формате закодировать сообщение.

■ **spec.socket.encoding.cef**

объект

Поля конфигурации, специфичные для CEF. Применяется только в том случае, если для кодирования установлено значение `CEF`.

■ **spec.socket.encoding.cef.deviceProduct**

строка

По умолчанию: `log-shipper-agent`

■ **spec.socket.encoding.cef.deviceVendor**

строка

По умолчанию: `Deckhouse`

■ **spec.socket.encoding.cef.deviceVersion**

строка

По умолчанию: `1`

■ **spec.socket.encoding.codec**

строка

Доступные форматы для кодирования.

По умолчанию: `JSON`

Допустимые значения: `Text`, `JSON`, `Syslog`, `CEF`, `GELF`

■ **spec.socket.mode**

строка

Обязательный параметр

Допустимые значения: `TCP`, `UDP`

■ **spec.socket.tcp**

объект

■ **spec.socket.tcp.tls**

объект

Настройки защищенного TLS-соединения.

■ **spec.socket.tcp.tls.caFile**

строка

Закодированный в Base64 сертификат CA в формате PEM.

■ **spec.socket.tcp.tls.clientCrt**

объект

Конфигурация клиентского сертификата.

■ **spec.socket.tcp.tls.clientCrt.crtFile**

строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

■ **spec.socket.tcp.tls.clientCrt.keyFile**

строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

■ **spec.socket.tcp.tls.clientCrt.keyPass**

строка

Закодированный в Base64 пароль для ключа.

■ **spec.socket.tcp.tls.secretRef**

объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (`ca.pem`), клиентский сертификат (`crt.pem`), приватный ключ (`key.pem`) и пароль ключа для клиентского

сертификата (keyPass) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.socket.tcp.tls.secretRef.name**
строка

Имя секрета с TLS-сертификатами.

- **spec.socket.tcp.verifyCertificate**
булевый

Проверка действия TLS-сертификата удаленного хоста.

Если установлено значение `false`, сертификат не проверяется в списках отзыва сертификатов.

По умолчанию: `true`

- **spec.socket.tcp.verifyHostname**
булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

◦ **spec.splunk**

объект

- **spec.splunk.endpoint**
строка

Обязательный параметр

Базовый URL для экземпляра Splunk.

Пример:

```
endpoint: https://http-inputs-hec.splunkcloud.com
```

- **spec.splunk.index**
строка

Имя индекса, куда будут сохранены данные. Это поле можно задать динамически.

- **spec.splunk.tls**
объект

Настройки защищенного TLS-соединения.

- **spec.splunk.tls.caFile**
строка

Закодированный в Base64 сертификат CA в формате PEM.

- **spec.splunk.tls.clientCert**
объект

Конфигурация клиентского сертификата.

- **spec.splunk.tls.clientCert.crtFile**
строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

- **spec.splunk.tls.clientCert.keyFile**
строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

- **spec.splunk.tls.clientCert.keyPass**
строка

Закодированный в Base64 пароль для ключа.

- **spec.splunk.tls.secretRef**
объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (ca.pem), клиентский сертификат (crt.pem), приватный ключ (key.pem) и пароль ключа для клиентского сертификата (keyPass) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.splunk.tls.secretRef.name**
строка

Имя секрета с TLS-сертификатами.

- **spec.splunk.tls.verifyCertificate**
булевый

Проверка действия TLS-сертификата удаленного хоста.

По умолчанию: `true`

- **spec.splunk.tls.verifyHostname**
булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

- **spec.splunk.token**
строка

Обязательный параметр

Токен по умолчанию для Splunk HEC. Если токен не был передан через `metadata`, будет использовано значение из этого поля.

- **spec.transformations**
массив объектов

Упорядоченный список трансформаций, применяемых к каждому логу перед отправкой в хранилище. Элементы выполняются в порядке следования в массиве.

Если заданы и `extraLabels`, и `AddLabels`, сначала применяются `extraLabels`, затем `AddLabels`.

Доступные действия:

- **ReplaceKeys** — рекурсивно заменяет вхождения шаблона `source` на `target` в указанных путях к ярлыкам.
- **ParseMessage** — разбирает строку из поля `message` (формат в `sourceFormat`) и записывает результат в `targetLabel` (по умолчанию `.message`; значение `.` — слияние с корнем лога). Если разбор невозможен, исходное поле `message` не меняется.
- **DropLabels** — удаляет значение по каждому пути `label` или, если у записи задан `keepKeys`, оставляет в объекте по этому пути только перечисленные **дочерние** ключи.
- **AddLabels** — добавляет ярлыки (литералы, шаблоны `{{ .path }}`, опционально `when`). Выполняется после `extraLabels`; при совпадении имён ключей приоритет у `AddLabels`.
- **ReplaceValue** — заменяет подстроки в скалярных строковых полях по регулярному выражению `source`; в `target` допускаются ссылки на именованные группы `{{ group_name }}` (`replaceValue`).
- **spec.transformations.action**
строка

Тип трансформации; определяет, какой из блоков параметров используется.

Допустимые значения: `ReplaceKeys`, `ParseMessage`, `DropLabels`, `AddLabels`, `ReplaceValue`

- **spec.transformations.addLabels**
объект
 - **spec.transformations.addLabels.setLabels**
объект

Обязательный параметр

- **spec.transformations.addLabels.when**
массив строк

- **Элемент массива**
строка

Шаблон: `^[a-zA-Z0-9_\\s\\[\\](){}?*.^$\\-!:, '" | <> ~ # / @] + $`

- **spec.transformations.dropLabels**
объект

Удаляет поля по путям или сужает объекты: при заданном `keepKeys` у элемента в объекте по пути `label` остаются только выбранные дочерние ключи.

- **spec.transformations.dropLabels.labels**
массив строк

У каждой записи обязателен путь `label` в нотации Vector (с ведущей `.`, например `.pod_labels`) и опционально `keepKeys`. Если `keepKeys` не задан — удаляется всё значение (поддерево) по пути `label`. Если `keepKeys` задан — в объекте по пути `label` сохраняются только перечисленные **дочерние** имена ключей (без ведущей точки), остальные дочерние ключи удаляются.

- **Элемент массива**
строка

Шаблон: `^\\.[a-zA-Z0-9_\\[\\]\\\\.\\-]+ $`

- **spec.transformations.dropLabels.paths**
массив объектов

List of path entries matching v1alpha2 dropLabels.labels.

- **spec.transformations.dropLabels.paths.keepKeys**

массив строк

- **Элемент массива**
строка

Шаблон: `^[a-zA-Z0-9_.\- /]+$`

- **spec.transformations.dropLabels.paths.label**
строка

Обязательный параметр

Шаблон: `^\.[a-zA-Z0-9_\\[\]\- /]+$`

- **spec.transformations.parseMessage**
объект

Разбирает строку из поля `message` (формат задаётся в `sourceFormat`) и записывает результат в путь `targetLabel` в нотации Vector с ведущей `.` (по умолчанию `.message`; значение `.` — слияние распаршенного объекта с корнем логга). Если разбор невозможен, исходное поле `message` не меняется.

- **spec.transformations.parseMessage.json**
объект
 - **spec.transformations.parseMessage.json.depth**
целочисленный

Ограничение глубины при разборе JSON (`max_depth` в Vector).

Допустимые значения: `1 <= X <= 128`

- **spec.transformations.parseMessage.sourceFormat**
строка

Обязательный параметр

Формат исходной строки в `message` для преобразования в объект (String, JSON, Klog, SysLog, CLF, Logfmt).

Допустимые значения: `String, JSON, Klog, SysLog, CLF, Logfmt`

- **spec.transformations.parseMessage.string**
объект

- **spec.transformations.parseMessage.string.regex**
строка

Регулярное выражение с именованными группами; требуется `setLabels`.

Шаблон: `^[a-zA-Z0-9_\\s\\[\\](){}?*.^$\\-=:!:',\"|<>~#/@]+&`

- **spec.transformations.parseMessage.string.setLabels**
объект

Сопоставление ключей результата литералам или шаблонам `{{ имя_группы }}`.

- **spec.transformations.parseMessage.string.targetField**
строка

Target field where the structured message is placed.

Шаблон: `^[a-zA-Z0-9_\\[\]\- /]+$`

- **spec.transformations.parseMessage.targetLabel**
строка

Путь назначения для распаршенного значения (по умолчанию `.message`). Значение `.` — слияние с корнем логга. Не может быть `.parsed_data`.

По умолчанию: `.message`

Шаблон: `^(\\.|\\.[a-zA-Z0-9_\\[\\]\\[\- /:']+$`

- **spec.transformations.replaceKeys**
объект

Рекурсивно заменяет вхождения шаблона `source` на значение `target` в перечисленных путях к ярлыкам (`labels`). Пути задаются в нотации Vector с ведущей `.` у каждого элемента списка.

- **spec.transformations.replaceKeys.labels**
массив строк

Обязательный параметр

Пути к ярлыкам, в которых выполняется замена (поле `labels`). Каждый элемент должен начинаться с `.` (нотация путей Vector, например `.pod_labels`, `.parsed_data`).

- **Элемент массива**
строка

Шаблон: `^\.[a-zA-Z0-9_\\[\]\- /]+$`

- **spec.transformations.replaceKeys.source**
строка

Обязательный параметр

Шаблон для поиска совпадений. Может быть статической строкой или регулярным выражением.

- **spec.transformations.replaceKeys.target**
строка

Значение, на которое будут заменены все совпадения шаблона.

По умолчанию:

- **spec.transformations.replaceValue**

объект

- **spec.transformations.replaceValue.labels**
массив строк

Обязательный параметр

- **Элемент массива**
строка

Шаблон: `^\.[a-zA-Z0-9_\\[\]\.\-"/:']+$`

- **spec.transformations.replaceValue.source**
строка

Обязательный параметр

Шаблон: `^[a-zA-Z0-9_\\s\\[\\](){}?*.^$\\-=:','" |<>~#/@]+$`

- **spec.transformations.replaceValue.target**
строка

Обязательный параметр

- **spec.type**
строка

Возможные бэкенды для сохранения логов.

Допустимые значения: `Loki`, `Elasticsearch`, `Logstash`, `Vector`, `Kafka`, `Splunk`, `Socket`

- **spec.vector**
объект

- **spec.vector.endpoint**
строка

Обязательный параметр

Адрес для подключения к Vector. Для общения между экземплярами должен использоваться API v2.

Шаблон: `^(.+):([0-9]{1,5})$`

- **spec.vector.tls**
объект

Настройки защищенного TLS-соединения.

- **spec.vector.tls.caFile**
строка

Закодированный в Base64 сертификат CA в формате PEM.

- **spec.vector.tls.clientCrt**
объект

Конфигурация клиентского сертификата.

- **spec.vector.tls.clientCrt.crtFile**
строка

Обязательный параметр

Закодированный в Base64 сертификат в формате PEM.

Также необходимо указать ключ в параметре `keyFile`.

- **spec.vector.tls.clientCrt.keyFile**
строка

Обязательный параметр

Закодированный в Base64 ключ в формате PEM.

Также необходимо указать сертификат в параметре `crtFile`.

- **spec.vector.tls.clientCrt.keyPass**
строка

Закодированный в Base64 пароль для ключа.

- **spec.vector.tls.secretRef**
объект

Ссылка на секрет Kubernetes, содержащий сертификат CA (ca.pem), клиентский сертификат

(`crt.pem`), приватный ключ (`key.pem`) и пароль ключа для клиентского сертификата (`keyPass`) в формате PEM, закодированном в Base64. Настройки TLS будут переопределены значениями из секрета. Секрет должен находиться в пространстве имён `d8-log-shipper` и иметь лейбл `log-shipper.deckhouse.io/watch-secret: true`.

- **spec.vector.tls.secretRef.name**

строка

Имя секрета с TLS-сертификатами.

- **spec.vector.tls.verifyCertificate**

булевый

Проверка действия TLS-сертификата удаленного хоста.

По умолчанию: `true`

- **spec.vector.tls.verifyHostname**

булевый

Проверка соответствия имени удаленного хоста и имени, указанного в TLS-сертификате удаленного хоста.

По умолчанию: `true`

ClusterLoggingConfig

Scope: Cluster

- `v1alpha2`
- `v1alpha1`

Описывает источник логов в log-pipeline.

Каждый ресурс `ClusterLoggingConfig` описывает правила сбора логов из кластера.

- **spec**
объект

Обязательный параметр

- **spec.destinationRefs**

массив строк

Обязательный параметр

Массив имен ресурса `ClusterLogDestination`, с которыми будет работать этот источник логов.

Поля с числовыми и булевыми типами будут преобразованы в строки.

- **spec.file**

объект

Описывает правило сбора логов из файлов на узле.

- **spec.file.exclude**

массив строк

Список шаблонов путей к файлам, которые нужно исключить при сборе логов.

Поддерживаются wildcards.

Примеры:

```
exclude: "/var/log/nginx/error.log"
```

```
exclude: "/var/log/audit.log"
```

- **spec.file.include**

массив строк

Список шаблонов путей к файлам для сбора логов.

Поддерживаются wildcards.

Примеры:

```
include: "/var/log/*.log"
```

```
include: "/var/log/nginx/*.log"
```

- **spec.file.lineDelimiter**

строка

Символ новой строки, который использовать при парсинге логов.

Пример:

```
lineDelimiter: "\\r\\n"
```

- **spec.kubernetesPods**
объект

Описывает правило сбора логов из подов кластера.

- **spec.kubernetesPods.keepDeletedFilesOpenedFor**
строка

Указывает время, в течение которого удаленные файлы будут оставаться открытыми для чтения. Vector будет сохранять метаданные подов в течение этого времени, чтобы читать логи из удаленных подов. Эта опция полезна в случаях недоступности хранилища логов или сетевого разделения. Vector будет держать файлы логов открытыми до их окончательной отправки в пункт назначения.

Применение опции может повлиять на потребление ресурсов Vector и также быть причиной нехватки места на диске из-за удаленных файлов с логами. Используйте с осторожностью.

Формат - строка, содержащая единицу времени в часах и минутах: 30m, 1h, 2h30m, 24h.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.kubernetesPods.labelSelector**
объект

Задаёт фильтр по лейблам пода.

Подробнее про выражения можно узнать в документации Kubernetes.

- **spec.kubernetesPods.labelSelector.matchExpressions**
массив объектов

Список выражений для лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.kubernetesPods.labelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.kubernetesPods.labelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: In, NotIn, Exists, DoesNotExist

- **spec.kubernetesPods.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Длина: 1..63

- **spec.kubernetesPods.labelSelector.matchLabels**
объект

Список лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchLabels:
foo: bar
baz: who
```

- **spec.kubernetesPods.namespaceSelector**
объект

Задаёт фильтр по неймспейсам.

Фильтр использует параметр `labelSelector` для определения неймспейсов, из которых должны собираться логи.

- **spec.kubernetesPods.namespaceSelector.labelSelector**
объект

Фильтр по лейблам неймспейсов, с которых нужно собирать логи.

Подробнее про выражения можно узнать в документации Kubernetes.

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions**
массив объектов

Список выражений для лейблов, которые должен иметь неймспейс, чтобы подпадать под условие фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.key**
строка

Обязательный параметр

Имя лейбла.

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.operator**
строка

Обязательный параметр

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Длина: 1..63

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchLabels**
объект

Список лейблов, которые должен иметь неймспейс, чтобы подпадать под условие фильтра.

Пример:

```
matchLabels:
foo: bar
baz: who
```

- **spec.labelFilter**
массив объектов

Список правил для фильтрации логов по их лейблам метаданных.

Пример:

```
labelFilter:
- field: container
  operator: In
  values:
  - nginx
- field: pod_labels.tier
  operator: Regex
  values:
  - prod-.+
  - stage-.+
```

- **spec.labelFilter.field**
строка

Обязательный параметр

Имя лейбла для использования в фильтре.

Не должно быть пустым.

Шаблон: `.+`

■ **spec.labelFilter.operator**

строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- **In** — ищет строку или элемент в массиве;
- **NotIn** — является инверсией оператора **In**;
- **Regex** — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- **NotRegex** — является инверсией оператора **Regex** (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- **Exists** — проверяет наличие поля и пропускает логи, только если поле есть;
- **DoesNotExist** — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: **In**, **NotIn**, **Regex**, **NotRegex**, **Exists**, **DoesNotExist**

■ **spec.labelFilter.values**

массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций **Exists** и **DoesNotExist**.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

○ **spec.logFilter**

массив объектов

Список фильтров для логов, которые применяются к полям сообщения в формате JSON.

Только логи, подпадающие под правила, будут сохранены в хранилище.

Пример:

```
logFilter:
- field: tier
  operator: Exists
- field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
- field: bar
  operator: Regex
  values:
  - "^abc"
  - "^\\d.+$"
```

■ **spec.logFilter.field**

строка

Обязательный параметр

Имя поля для фильтрации. Должно быть пустым для логов не в JSON-формате.

■ **spec.logFilter.operator**

строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- **In** — ищет строку или элемент в массиве;
- **NotIn** — является инверсией оператора **In**;
- **Regex** — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- **NotRegex** — является инверсией оператора **Regex** (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- **Exists** — проверяет наличие поля и пропускает логи, только если поле есть.
- **DoesNotExist** — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: **In**, **NotIn**, **Regex**, **NotRegex**, **Exists**, **DoesNotExist**

■ **spec.logFilter.values**

массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций **Exists** и **DoesNotExist**.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

- **spec.multilineParser**
объект

Список паттернов для определения многострочных логов.

- **spec.multilineParser.custom**
объект

Правила парсинга многострочных логов для парсера `Custom`.

- **spec.multilineParser.custom.endsWhen**
объект

Условие, при котором находится последняя строка многострочного лога.

- **spec.multilineParser.custom.endsWhen.notRegex**
строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

- **spec.multilineParser.custom.endsWhen.regex**
строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

- **spec.multilineParser.custom.startsWhen**
объект

Условие, при котором находится первая строка многострочного лога.

- **spec.multilineParser.custom.startsWhen.notRegex**
строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

- **spec.multilineParser.custom.startsWhen.regex**
строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

- **spec.multilineParser.type**
строка

Обязательный параметр

Типы парсеров:

- `None` — не парсить логи;
- `General` — парсер, который пытается сливать логи, у которых в начале сообщения стоит пробел или табуляция;
- `Backslash` — парсер, который парсит многострочные логи в SHELL-формате с обратным слэшем у строк одного сообщения;
- `LogWithTime` — парсер, который ожидает, что любое новое сообщение начинается с временного лейбла;
- `MultilineJSON` — простой парсер JSON-логов, который предполагает, что новое сообщение начинается с символа `{`;
- `Custom` - парсер, который парсит многострочные логи в указанном пользователем в `spec.multilineParser.custom` формате.

По умолчанию: `None`

Допустимые значения: `None`, `General`, `Backslash`, `LogWithTime`, `MultilineJSON`, `Custom`

- **spec.type**
строка

Обязательный параметр

Возможные источники логов.

`KubernetesPods` собирает логи с подов.

`File` позволяет читать локальные файлы, доступные на узле.

Допустимые значения: `KubernetesPods`, `File`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описывает источник логов в log-pipeline.

Каждый ресурс `ClusterLoggingConfig` описывает правила сбора логов из кластера.

- **spec**
объект

Обязательный параметр

- **spec.destinationRefs**
массив строк

Обязательный параметр

Массив имен ресурса `ClusterLogDestination`, с которыми будет работать этот источник логов.

Поля с числовыми и булевыми типами будут преобразованы в строки.

- **spec.file**
объект

Описывает правило сбора логов из файлов на узле.

- **spec.file.exclude**
массив строк

Список шаблонов путей к файлам, которые нужно исключить при сборе логов.

Поддерживаются wildcards.

Примеры:

```
exclude: "/var/log/nginx/error.log"
```

```
exclude: "/var/log/audit.log"
```

- **spec.file.include**
массив строк

Список шаблонов путей к файлам для сбора логов.

Поддерживаются wildcards.

Примеры:

```
include: "/var/log/*.log"
```

```
include: "/var/log/nginx/*.log"
```

- **spec.file.lineDelimiter**
строка

Символ новой строки, который использовать при парсинге логов.

Пример:

```
lineDelimiter: "\\r\\n"
```

- **spec.kubernetesPods**
объект

Описывает правило сбора логов из подов кластера.

- **spec.kubernetesPods.keepDeletedFilesOpenedFor**
строка

Указывает время, в течение которого удаленные файлы будут оставаться открытыми для чтения.

Vector будет сохранять метаданные подов в течение этого времени, чтобы читать логи из удаленных подов. Эта опция полезна в случаях недоступности хранилища логов или сетевого разделения.

Vector будет держать файлы логов открытыми до их окончательной отправки в пункт назначения.

Применение опции может повлиять на потребление ресурсов Vector и также быть причиной нехватки места на диске из-за удаленных файлов с логами. Используйте с осторожностью.

Формат - строка, содержащая единицу времени в часах и минутах: 30m, 1h, 2h30m, 24h.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.kubernetesPods.labelSelector**
объект

Задаёт фильтр по лейблам пода.

Подробнее про выражения можно узнать в документации Kubernetes.

- **spec.kubernetesPods.labelSelector.matchExpressions**
массив объектов

Список выражений для лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.kubernetesPods.labelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.kubernetesPods.labelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.kubernetesPods.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Длина: `1..63`

- **spec.kubernetesPods.labelSelector.matchLabels**
объект

Список лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.kubernetesPods.namespaceSelector**
объект

Задаёт фильтр по неймспейсам.

Фильтр может использовать какой-то один способ задания условия из трех доступных (параметры `matchNames`, `excludeNames`, `labelSelector`).

- **spec.kubernetesPods.namespaceSelector.excludeNames**
массив строк

Список неймспейсов, с подов которого нужно исключить сбор логов.

- **spec.kubernetesPods.namespaceSelector.labelSelector**
объект

Фильтр по лейблам неймспейсов, с которых нужно собирать логи.

Подробнее про выражения можно узнать в документации Kubernetes.

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions**
массив объектов

Список выражений для лейблов, которые должен иметь неймспейс, чтобы подпадать под условие фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.operator**

строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**

строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Длина: 1..63

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchLabels**
объект

Список лейблов, которые должен иметь неймспейс, чтобы подпадать под условие фильтра.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.kubernetesPods.namespaceSelector.matchNames**
массив строк

Список неймспейсов, с подов которого нужно собирать логи.

◦ **spec.labelFilter**

массив объектов

Список правил для фильтрации логов по их лейблам метаданных.

Пример:

```
labelFilter:
- field: container
  operator: In
  values:
  - nginx
- field: pod_labels.tier
  operator: Regex
  values:
  - prod-.+
  - stage-.+
```

- **spec.labelFilter.field**
строка

Обязательный параметр

Имя лейбла для использования в фильтре.

Не должно быть пустым.

Шаблон: `.*`

- **spec.labelFilter.operator**
строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- `In` — ищет строку или элемент в массиве;
- `NotIn` — является инверсией оператора `In`;
- `Regex` — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- `NotRegex` — является инверсией оператора `Regex` (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- `Exists` — проверяет наличие поля и пропускает логи, только если поле есть;
- `DoesNotExist` — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.labelFilter.values**
массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций `Exists` и `DoesNotExist`.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

- **spec.logFilter**
массив объектов

Список фильтров для логов, которые применяются к полям сообщения в формате JSON.

Только логи, подпадающие под правила, будут сохранены в хранилище.

Пример:

```
logFilter:
- field: tier
  operator: Exists
- field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
- field: bar
  operator: Regex
  values:
  - "^abc"
  - "\\d.+\\$"
```

- **spec.logFilter.field**
строка

Обязательный параметр

Имя поля для фильтрации. Должно быть пустым для логов не в JSON-формате.

- **spec.logFilter.operator**
строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- **In** — ищет строку или элемент в массиве;
- **NotIn** — является инверсией оператора **In**;
- **Regex** — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- **NotRegex** — является инверсией оператора **Regex** (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- **Exists** — проверяет наличие поля и пропускает логи, только если поле есть.
- **DoesNotExist** — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: **In**, **NotIn**, **Regex**, **NotRegex**, **Exists**, **DoesNotExist**

- **spec.logFilter.values**
массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций **Exists** и **DoesNotExist**.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

- **spec.multilineParser**
объект

Список паттернов для определения многострочных логов.

- **spec.multilineParser.custom**
объект

Правила парсинга многострочных логов для парсера **Custom**.

- **spec.multilineParser.custom.endsWhen**
объект

Условие, при котором находится последняя строка многострокового лога.

- **spec.multilineParser.custom.endsWhen.notRegex**
строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

- **spec.multilineParser.custom.endsWhen.regex**
строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

- **spec.multilineParser.custom.startsWhen**
объект

Условие, при котором находится первая строка многострокового лога.

- **spec.multilineParser.custom.startsWhen.notRegex**
строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

- **spec.multilineParser.custom.startsWhen.regex**
строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

- **spec.multilineParser.type**
строка

Обязательный параметр

Типы парсеров:

- **None** — не парсить логи;
- **General** — парсер, который пытается сливать логи, у которых в начале сообщения стоит пробел или табуляция;
- **Backslash** — парсер, который парсит многострочные логи в SHELL-формате с обратным слэшем у строк одного сообщения;
- **LogWithTime** — парсер, который ожидает, что любое новое сообщение начинается с временного лейбла;
- **MultilineJSON** — простой парсер JSON-логов, который предполагает, что новое сообщение начинается с символа {;
- **Custom** - парсер, который парсит многострочные логи в указанном пользователем в `spec.multilineParser.custom` формате.

По умолчанию: **None**

Допустимые значения: **None**, **General**, **Backslash**, **LogWithTime**, **MultilineJSON**, **Custom**

- **spec.type**
строка

Обязательный параметр

Возможные источники логов.

KubernetesPods собирает логи с подов.

File позволяет читать локальные файлы, доступные на узле.

Допустимые значения: **KubernetesPods**, **File**

PodLoggingConfig

Scope: Namespaced

Version: v1alpha1

Ресурс для описания источника логов в log-pipeline.

Каждый ресурс **PodLoggingConfig** описывает правила сбора логов из указанного namespace.

- **spec**
объект

Обязательный параметр

- **spec.clusterDestinationRefs**
массив строк

Обязательный параметр

Список бэкэндов хранения (CRD **ClusterLogDestination**), в которые будет отправлено сообщение.

- **spec.keepDeletedFilesOpenedFor**
строка

Указывает время, в течение которого удаленные файлы будут оставаться открытыми для чтения. Vector будет сохранять метаданные подов в течение этого времени, чтобы читать логи из удаленных подов. Эта опция полезна в случаях недоступности хранилища логов или сетевого разделения. Vector будет держать файлы логов открытыми до их окончательной отправки в пункт назначения.

Применение опции может повлиять на потребление ресурсов Vector и также быть причиной нехватки места на диске из-за удаленных файлов с логами. Используйте с осторожностью.

Формат - строка, содержащая единицу времени в часах и минутах: 30m, 1h, 2h30m, 24h.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.labelFilter**
массив объектов

Список правил для фильтрации логов по их лейблам метаданных.

Пример:

```
labelFilter:
- field: container
  operator: In
  values:
  - nginx
- field: pod_labels.tier
  operator: Regex
  values:
  - prod-.*
  - stage-.*
- field: message
  operator: Regex
  values:
  - ".*search_text.*"
```

■ **spec.labelFilter.field**

строка

Обязательный параметр

Имя лейбла для использования в фильтре.

Не должно быть пустым.

Шаблон: `.*`

■ **spec.labelFilter.operator**

строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- **In** — ищет строку или элемент в массиве;
- **NotIn** — является инверсией оператора **In**;
- **Regex** — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- **NotRegex** — является инверсией оператора **Regex** (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- **Exists** — проверяет наличие поля и пропускает логи, только если поле есть;
- **DoesNotExist** — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

■ **spec.labelFilter.values**

массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций `Exists` и `DoesNotExist`.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

○ **spec.labelSelector**

объект

Задаёт фильтр по лейблам пода.

Подробнее про выражения можно узнать в документации.

■ **spec.labelSelector.matchExpressions**

массив объектов

Список выражений для лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

■ **spec.labelSelector.matchExpressions.key**

строка

Имя лейбла.

■ **spec.labelSelector.matchExpressions.operator**

строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

■ **spec.labelSelector.matchExpressions.values**

массив строк

Значение лейбла.

- **Элемент массива**

строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: 1..63

- **spec.selector.matchLabels**

объект

Список лейблов, которые должен иметь под, чтобы подпадать под условие фильтра.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.logFilter**

массив объектов

Список фильтров для логов, которые применяются к полям сообщения в формате JSON.

Только логи, подпавшие под правила, будут сохранены в хранилище.

Пример:

```
logFilter:
- field: tier
  operator: Exists
- field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
- field: bar
  operator: Regex
  values:
  - "^abc"
  - "^\\d.+$"
```

- **spec.logFilter.field**

строка

Обязательный параметр

Имя поля для фильтрации. Должно быть пустым для логов не в JSON-формате.

- **spec.logFilter.operator**

строка

Обязательный параметр

Оператор, который можно применить для фильтрации:

- `In` — ищет строку или элемент в массиве;
- `NotIn` — является инверсией оператора `In`;
- `Regex` — пытается проверить строку в поле с использованием регулярного выражения (только логи, в которых есть поля, подпадающие под регулярное выражение, пройдут в хранилище);
- `NotRegex` — является инверсией оператора `Regex` (в хранилище попадут логи, в которых нет поля или же оно не подпадает под регулярное выражение);
- `Exists` — проверяет наличие поля и пропускает логи, только если поле есть;
- `DoesNotExist` — проверяет наличие поля и пропускает логи, только если поле отсутствует.

Допустимые значения: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.logFilter.values**

массив

Массив значений или регулярных выражений для соответствующих операций. Не работает для операций `Exists` и `DoesNotExist`.

Можно использовать целые числа или строки. Поля с числами с плавающей запятой и поля логического типа будут преобразованы в строки при сравнении.

- **spec.multilineParser**

объект

Список паттернов для определения многострочных логов.

- **spec.multilineParser.custom**

объект

Правила парсинга многострочных логов для парсера `Custom`.

■ **spec.multilineParser.custom.endsWith**

объект

Условие, при котором находится последняя строка многострокового лога.

■ **spec.multilineParser.custom.endsWith.notRegex**

строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

■ **spec.multilineParser.custom.endsWith.regex**

строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

■ **spec.multilineParser.custom.startsWith**

объект

Условие, при котором находится первая строка многострокового лога.

■ **spec.multilineParser.custom.startsWith.notRegex**

строка

Регулярное выражение, которое считает мэтчем строки, НЕ попавшие в него.

■ **spec.multilineParser.custom.startsWith.regex**

строка

Регулярное выражение, которое считает мэтчем строки, попавшие в него.

■ **spec.multilineParser.type**

строка

Обязательный параметр

Типы парсеров:

- `None` — не парсить логи;
- `General` — парсер, который пытается сливать логи, у которых в начале сообщения стоит пробел или табуляция;
- `Backslash` — парсер, который парсит многострочные логи в SHELL-формате с обратным слэшем у строк одного сообщения;
- `LogWithTime` — парсер, который ожидает, что любое новое сообщение начинается с временного лейбла;
- `MultilineJSON` — простой парсер JSON-логов, который предполагает, что новое сообщение начинается с символа `{`;
- `Custom` - парсер, который парсит многострочные логи в указанном пользователем в `spec.multilineParser.custom` формате.

По умолчанию: `None`

Допустимые значения: `None`, `General`, `Backslash`, `LogWithTime`, `MultilineJSON`, `Custom`

The log-shipper module: FAQ

Как добавить авторизацию в ресурс `ClusterLogDestination`?

Чтобы добавить параметры авторизации в ресурс `ClusterLogDestination`, необходимо:

- изменить протокол подключения к Loki на HTTPS;
- добавить секцию `auth`, в которой:
 - параметр `strategy` установить в `Bearer`;
 - в параметре `token` указать токен `log-shipper-token` из неймспейса `d8-log-shipper`.

Пример:

- Ресурс `ClusterLogDestination` без авторизации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki
spec:
  type: Loki
  loki:
    endpoint: "http://loki.d8-monitoring:3100"
```

- Получите токен `log-shipper-token` из неймспейса `d8-log-shipper`:

```
d8 k -n d8-log-shipper get secret log-shipper-token -o jsonpath='{.data.token}' | base64 -d
```

- Ресурс `ClusterLogDestination` с авторизацией:


```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki
spec:
  type: Loki
  loki:
    endpoint: "https://loki.d8-monitoring:3100"
    auth:
      strategy: "Bearer"
      token: <log-shipper-token>
    tls:
      verifyHostname: false
      verifyCertificate: false

```

Расширенная конфигурация

Как включить debug-логи?

Включите параметр debug модуля.

Пример конфигурации модуля:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 1
  enabled: true
  settings:
    debug: true

```

После этого в логах вы найдете много полезной информации о HTTP-запросах, переиспользовании подключения, трассировку и т. д.

Как узнать больше о каналах передачи log'ов?

Для начала получите список подов на желаемом узле.

```
d8 k -n d8-log-shipper get pods -O wide | grep $node
```

Выберите нужный под и выполните команды напрямую из контейнера.

```
d8 k -n d8-log-shipper exec $pod -c vector -- ИМЯ_КОМАНДЫ
```

Все следующие команды предполагается запускать передавая их в контейнер.

Посмотреть каналы как graph

- Выполните команду `vector graph`, чтобы получить graph в формате DOT.
- Выставьте его в webgraphviz или другой похожий сервис, чтобы получить изображение.

Пример graph'а для одного канала передачи логов в формате ASCII:

```

+-----+
| d8_cluster_source_flant-integration-d8-logs |
+-----+
|
|
v
+-----+
| d8_tf_flant-integration-d8-logs_0 |
+-----+
|
|
v
+-----+
| d8_tf_flant-integration-d8-logs_1 |
+-----+
|
|
v
+-----+
| d8_cluster_sink_flant-integration-loki-storage |
+-----+

```

Отладка передачи данных

Существует команда `vector top`, которая поможет увидеть, как много трафика проходит через каждую стадию обработки логов.

Пример вывода команды:

```

-Vector-
http://127.0.0.1:8686/graphql | Sampling @ 500ms

-Components-

```

ID	Output	Kind	Type	Events In	Events Out	Bytes	Errors
d8_cluster_sink_flant	--	sink	loki	324,198 (--/s)	324,198 (--/s)	N/A	--
d8_cluster_source_fla	--	source	kubernetes_log	324,198 (--/s)	324,198 (--/s)	1,099,955,306	--
d8_tf_flant-integrati	--	transform	remap	324,198 (--/s)	324,198 (--/s)	N/A	--
d8_tf_flant-integrati	--	transform	remap	324,198 (--/s)	324,198 (--/s)	N/A	--
internal_metrics	--	source	internal_metri	5,158,622 (--/s)	5,158,622 (--/s)	N/A	--
prometheus_sink	--	sink	prometheus	5,158,622 (--/s)	N/A	N/A	--

Получить необработанные образцы log'ов

Вы можете использовать команды `vector tap`, чтобы получить входные данные для любых компонентов каналов. Единственный аргумент, который нужно задать, — ID стадии обработки (также поддерживаются выборки в формате glob, например `cluster_logging_config/*`).

С помощью следующей команды вы сможете увидеть логи до применения правил трансформации (`cluster_logging_config/*` является первой стадией обработки согласно выводу команды `vector graph`):

```
vector tap 'cluster_logging_config/*'
```

Измененные логи, которые поступают на вход следующих в цепочке компонентов каналов:

```
vector tap 'transform/*'
```

Вы также можете использовать интерактивную консоль `vector vrl` для отладки VRL правил для изменения логов.

Пример программы с использованием VRL:

```

. = {"test1": "lynx", "test2": "fox"}
del(.test2)
.

```

Как добавить поддержку нового source/sink для log-shipper?

Vector для модуля `log-shipper` был собран с ограниченным набором функций (чтобы уменьшить время сборки и размер финального запускаемого файла).

Вы можете посмотреть весь список поддерживаемых функций, выполнив команду `vector list`.

Если необходимо добавить поддержку нового source/sink, сначала нужно добавить необходимую функцию в список поддерживаемых в Dockerfile.

Примеры

Чтение логов из всех подов кластера и направление их в Loki

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
    - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Чтобы направить логи в кластерный модуль loki укажите `d8-loki` в параметре destinationRefs ClusterLoggingConfig. Дополнительные примеры можно найти в документации.

Чтение логов подов из указанного namespace с указанным label и перенаправление одновременно

в Loki и Elasticsearch

Чтение логов подов из namespace `whispers` только с label `app=booking` и перенаправление одновременно в Loki и Elasticsearch:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: whispers-booking-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: "kubernetes.io/metadata.name"
            operator: In
            values: [whispers]
        labelSelector:
          matchLabels:
            app: booking
    destinationRefs:
      - loki-storage
      - es-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Создание source в namespace и чтение логов всех подов в этом NS с направлением их в Loki

Следующий pipeline создает source в namespace `test-whispers`, читает логи всех подов в этом NS и пишет их в Loki:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: whispers-logs
  namespace: tests-whispers
spec:
  clusterDestinationRefs:
    - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Чтение только подов в указанном namespace и с определенным label

Пример чтения только подов, имеющих label `app=booking`, в namespace `test-whispers`:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: whispers-logs
  namespace: tests-whispers
spec:
  labelSelector:
    matchLabels:
      app: booking
  clusterDestinationRefs:
    - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Переход с Promtail на Log-Shipper

В ранее используемом URL Loki требуется убрать путь `/loki/api/v1/push`.

Vector сам добавит этот путь при работе с Loki.

Работа с Grafana Cloud

Данная документация подразумевает, что у вас уже создан ключ API.

Для начала вам потребуется закодировать в base64 ваш токен доступа к Grafana Cloud.

URL: `https://logs-prod-eu-west-0.grafana.net`

Basic Auth: ✓ (checked)

User: 

Password: Your Grafana.com API Key. [Generate now](#)

```
echo -n "<YOUR-GRAFANA-CLOUD-TOKEN>" | base64 -w0
```

Затем нужно создать **ClusterLogDestination**

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  loki:
    auth:
      password: PFlPVVItR1JBRkF0QUNMT1VELVRPS0VOPg==
      strategy: Basic
      user: "<YOUR-GRAFANA-CLOUD-USER>"
    endpoint: <YOUR-GRAFANA-CLOUD-URL> # Например https://logs-prod-us-central1.grafana.net или
https://logs-prod-eu-west-0.grafana.net
    type: Loki
```

Теперь можно создать PodLogginConfig или ClusterPodLoggingConfig и отправлять логи в **Grafana Cloud**.

Добавление Loki в Deckhouse Grafana

Вы можете работать с Loki из встроенной в Deckhouse Grafana. Достаточно добавить **GrafanaAdditionalDatasource**.

```

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: loki
spec:
  access: Proxy
  basicAuth: false
  jsonData:
    maxLines: 5000
    timeInterval: 30s
  type: loki
  url: http://loki.loki:3100

```

Поддержка Elasticsearch < 6.X

Для Elasticsearch < 6.0 нужно включить поддержку doc_type индексов. Сделать это можно следующим образом:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    docType: "myDocType" # Укажите значение здесь. Оно не должно начинаться с '_'.
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmlvOICluCg==

```

Шаблон индекса для Elasticsearch

Существует возможность отправлять сообщения в определенные индексы на основе метаданных с помощью шаблонов индексов:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s-{{ namespace }}-%F"

```

В приведенном выше примере для каждого неймспейса Kubernetes будет создан свой индекс в Elasticsearch.

Эта функция также хорошо работает в комбинации с `extraLabels`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s-{{ service }}-{{ namespace }}-%F"
  extraLabels:
    service: "{{ service_name }}"

```

1. Если сообщение имеет формат JSON, поле `service_name` этого документа JSON перемещается на уровень метаданных.
2. Новое поле метаданных `service` используется в шаблоне индекса.

Пример интеграции со Splunk

Существует возможность отсылать события из Deckhouse в Splunk.

1. Endpoint должен быть таким же, как имя вашего экземпляра Splunk с портом `8088` и без указания пути, например `https://prd-p-xxxxxx.splunkcloud.com:8088`.
2. Чтобы добавить token для доступа, откройте пункт меню `Setting -> Data inputs`, добавьте новый `HTTP Event Collector` и скопируйте token.

3. Укажите индекс Splunk для хранения логов, например `logs`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: splunk
spec:
  type: Splunk
  splunk:
    endpoint: https://prd-p-xxxxxx.splunkcloud.com:8088
    token: xxxx-xxxx-xxxx
    index: logs
    tls:
      verifyCertificate: false
      verifyHostname: false
```

i `destination` не поддерживает лейблы пода для индексирования. Рассмотрите возможность добавления нужных лейблов с помощью опции `extraLabels`.

```
extraLabels:
  pod_label_app: '{{ pod_labels.app }}'
```

Простой пример Logstash

Чтобы отправлять логи в Logstash, на стороне Logstash должен быть настроен входящий поток `tcp` и его кодек должен быть `json`.

Пример минимальной конфигурации Logstash:

```
input {
  tcp {
    port => 12345
    codec => json
  }
}
output {
  stdout { codec => json }
}
```

Пример манифеста `ClusterLogDestination`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: logstash
spec:
  type: Logstash
  logstash:
    endpoint: logstash.default:12345
```

Syslog

Следующий пример показывает, как отправлять сообщения через сокет по протоколу TCP в формате syslog:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: rsyslog
spec:
  type: Socket
  socket:
    mode: TCP
    address: 192.168.0.1:3000
    encoding:
      codec: Syslog
  extraLabels:
    syslog.severity: "alert"
    # поле request_id должно присутствовать в сообщении
    syslog.message_id: '{{ request_id }}'
```

Пример интеграции с Graylog

Убедитесь, что в Graylog настроен входящий поток для приема сообщений по протоколу TCP на указанном порту. Пример манифеста для интеграции с Graylog:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: test-socket2-dest
spec:
  type: Socket
  socket:
    address: graylog.svc.cluster.local:9200
    mode: TCP
    encoding:
      codec: GELF
```

Логи в CEF формате

Существует способ формировать логи в формате CEF, используя `codec: CEF`, с переопределением `cef.name` и `cef.severity` по значениям из поля `message` (лога приложения) в формате JSON.

В примере ниже `app` и `log_level` это ключи содержащие значения для переопределения:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: siem-kafka
spec:
  extraLabels:
    cef.name: '{{ app }}'
    cef.severity: '{{ log_level }}'
  type: Kafka
  kafka:
    bootstrapServers:
      - my-cluster-kafka-brokers.kafka:9092
    encoding:
      codec: CEF
    tls:
      verifyCertificate: false
      verifyHostname: true
    topic: logs
```

Так же можно вручную задать свои значения:

```
extraLabels:
  cef.name: 'TestName'
  cef.severity: '1'
```

Сбор событий Kubernetes

События Kubernetes могут быть собраны log-shipper'ом, если `events-exporter` включен в настройках модуля `extended-monitoring`.

Включите `events-exporter`, изменив параметры модуля `extended-monitoring`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: extended-monitoring
spec:
  version: 1
  settings:
    events:
      exporterEnabled: true
```

Выложите в кластер следующий `ClusterLoggingConfig`, чтобы собирать сообщения с пода `events-exporter`:

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubernetes-events
spec:
  type: KubernetesPods
  kubernetesPods:
    labelSelector:
      matchLabels:
        app: events-exporter
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: "kubernetes.io/metadata.name"
            operator: In
            values: [d8-monitoring]
  destinationRefs:
    - loki-storage

```

Фильтрация логов

Пользователи могут фильтровать логи, используя следующие фильтры:

- `labelFilter` — применяется к метаданным, например имени контейнера (`container`), неймспейсу (`namespace`) или имени пода (`pod_name`);
- `logFilter` — применяется к полям самого сообщения, если оно в JSON-формате.

Сборка логов только для контейнера `nginx`

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: nginx-logs
spec:
  type: KubernetesPods
  labelFilter:
    - field: container
      operator: In
      values: [nginx]
  destinationRefs:
    - loki-storage

```

Сборка логов без строки, содержащей `GET /status" 200`

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
    - loki-storage
  labelFilter:
    - field: message
      operator: NotRegex
      values:
        - .*GET /status" 200$

```

Аудит событий kubelet'a


```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: ["kubelet.*"]
  destinationRefs:
    - loki-storage

```

Системные логи Deckhouse

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: system-logs
spec:
  type: File
  file:
    include:
      - /var/log/syslog
  labelFilter:
    - field: message
      operator: Regex
      values:
        - .*d8-kubelet-forker.*
        - .*containerd.*
        - .*bashible.*
        - .*kernel.*
  destinationRefs:
    - loki-storage

```

i Если вам нужны только логи одного пода или малой группы подов, постарайтесь использовать настройки `kubernetesPods`, чтобы сузить количество читаемых файлов. Фильтры необходимы только для высокогранулярной настройки.

Преобразование логов

В ресурсе `ClusterLogDestination` можно задать цепочку трансформаций по порядку. Поддерживаются следующие действия: `ParseMessage`, `ReplaceKeys`, `ReplaceValue`, `AddLabels`, `DropLabels`.

Преобразование логов в структурированный объект

Вы можете использовать трансформацию `ParseMessage`, чтобы преобразовать строку в поле `message` в структурированный объект. Для формата `String` задаётся регулярное выражение с именованными группами и карта `setLabels`. Ключи результата сопоставляются с группами через шаблоны `{{ имя_группы }}`. При использовании нескольких трансформаций `ParseMessage`, преобразование строки должно выполняться последним.

В `parseMessage.targetLabel` нельзя задать путь `.parsed_data`.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: string-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: String
        string:
          regex: "^(?P<msg>.*)$"
          setLabels:
            msg: "{{ msg }}"

```

Пример изначальной записи в логге:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Результат преобразования:

```
{... "message": {
  "msg": "/docker-entrypoint.sh: Configuration complete; ready for start up"
}
}
```

Преобразование логов в формате Klog в структурированный объект

Вы можете использовать трансформацию `ParseMessage`, чтобы распарсить логи в формате Klog и преобразовать их в структурированный объект.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: klog-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: Klog
```

Пример изначальной записи в логе:

```
I0505 17:59:40.692994 28133 klog.go:70] hello from klog
```

Результат преобразования:

```
{... "message": {
  "file": "klog.go",
  "id": 28133,
  "level": "info",
  "line": 70,
  "message": "hello from klog",
  "timestamp": "2025-05-05T17:59:40.692994Z"
}
}
```

Преобразование логов в формате Syslog в структурированный объект

Вы можете использовать трансформацию `ParseMessage`, чтобы распарсить логи в формате Syslog и преобразовать их в структурированный объект.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: syslog-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: SysLog
```

Пример изначальной записи в логе:

```
<13>1 2020-03-13T20:45:38.119Z dynamicwireless.name non 2426 ID931 [exampleSDID@32473 iut="3" eventSource= "Application"
eventID="1011"] Try to override the THX port, maybe it will reboot the neural interface!
```

Результат преобразования:

```
{... "message": {
  "appname": "non",
  "exampleSDID@32473": {
    "eventID": "1011",
    "eventSource": "Application",
    "iut": "3"
  },
  "facility": "user",
  "hostname": "dynamicwireless.name",
  "message": "Try to override the THX port, maybe it will reboot the neural interface!",
  "msgid": "ID931",
  "procid": 2426,
  "severity": "notice",
  "timestamp": "2020-03-13T20:45:38.119Z",
  "version": 1
}}
```

Преобразование логов в формате CLF в структурированный объект

Вы можете использовать трансформацию `ParseMessage`, чтобы распарсить логи в формате CLF и преобразовать их в структурированный объект.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: clf-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: CLF
```

Пример изначальной записи в логе:

```
127.0.0.1 bob frank [10/Oct/2000:13:55:36 -0700] \"GET /apache_pb.gif HTTP/1.0\" 200 2326
```

Результат преобразования:

```
{... "message": {
  "host": "127.0.0.1",
  "identity": "bob",
  "message": "GET /apache_pb.gif HTTP/1.0",
  "method": "GET",
  "path": "/apache_pb.gif",
  "protocol": "HTTP/1.0",
  "size": 2326,
  "status": 200,
  "timestamp": "2000-10-10T20:55:36Z",
  "user": "frank"
}}
```

Преобразование логов в формате logfmt в структурированный объект

Вы можете использовать трансформацию `ParseMessage`, чтобы распарсить логи в формате Logfmt и преобразовать их в структурированный объект.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: logfmt-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: Logfmt
```

Пример изначальной записи в логе:

```
@timestamp=\"Sun Jan 10 16:47:39 EST 2021\" level=info msg=\"Stopping all fetchers\" tag#production=stopping_fetchers
id=ConsumerFetcherManager-1382721708341 module=kafka.consumer.ConsumerFetcherManager
```

Результат преобразования:

```
{... "message": {
  "@timestamp": "Sun Jan 10 16:47:39 EST 2021",
  "id": "ConsumerFetcherManager-1382721708341",
  "level": "info",
  "module": "kafka.consumer.ConsumerFetcherManager",
  "msg": "Stopping all fetchers",
  "tag#production": "stopping_fetchers"
}}
```

Парсинг JSON и уменьшение вложенности

Вы можете использовать трансформацию `ParseMessage`, чтобы парсить записи в логах в формате JSON. С помощью параметра `depth` можно контролировать глубину вложенности.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
        json:
          depth: 1
```

Пример изначальной записи в логе:

```
{"level" : { "severity": "info" }, "msg" : "fetching.module.release"}
```

Результат преобразования:

```
{... "message": {
  "level" : "{ \"severity\": \"info\" }",
  "msg" : "fetching.module.release"
}
}
```

Пример парсинга записей смешанных форматов в объект

Преобразование строки должно выполняться последним.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: ParseMessage
      parseMessage:
        sourceFormat: Klog
    - action: ParseMessage
      parseMessage:
        sourceFormat: String
        string:
          regex: "^(?P<text>.*)$"
          setLabels:
            text: "{{ text }}"
```

Пример изначальной записи в логе:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
{"level" : { "severity": "info" }, "msg" : "fetching.module.release"}
I0505 17:59:40.692994 28133 klog.go:70] hello from klog
```

Результат преобразования:

```
{... "message": {
  "text": "/docker-entrypoint.sh: Configuration complete; ready for start up"
}
}
{... "message": {
  "level" : "{ "severity": "info" }",
  "msg" : "fetching.module.release"
}
}
{... "message": {
  "file": "klog.go",
  "id": 28133,
  "level": "info",
  "line": 70,
  "message": "hello from klog",
  "timestamp": "2025-05-05T17:59:40.692994Z"
}
}
```

Замена лейблов

Вы можете использовать трансформацию `ReplaceKeys`, чтобы заменить `source` на `target` в заданных ключах лейблов.

{% alert level="info" %} Перед применением трансформации `ReplaceKeys` к полю `message` или его вложенным полям необходимо преобразовать запись лога в структурированный объект с помощью трансформации `ParseMessage`. {% endalert %}

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: replace-dot
spec:
  ...
  transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
```

Пример изначальной записи в логе:

```
{"msg" : "fetching.module.release"} # Лейбл пода pod.app=test
```

Результат преобразования:

```
{... "message": {
  "msg" : "fetching.module.release"
},
  "pod_labels": {
    "pod_app": "test"
  }
}
```

Замена подстрок в строковых полях

Трансформация `ReplaceValue` заменяет вхождения регулярного выражения `source` в скалярных строковых полях. В `target` можно задать литерал или шаблон с именованными группами `{{ имя }}` из `source`.

{% alert level="info" %} Если замена нужна для поля внутри структурированного `message`, сначала примените `ParseMessage`. {% endalert %}

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: redact-digits
spec:
  ...
  transformations:
    - action: ReplaceValue
      replaceValue:
        source: \d+
        target: REDACTED
      labels:
        - .message

```

Пример изначальной записи в логе:

```
user id 12345 connected from 10.0.0.1
```

Результат преобразования:

```
{... "message": "user id REDACTED connected from REDACTED"
}
```

Добавление лейблов из литералов и шаблонов

Трансформация `AddLabels` добавляет поля со статическими значениями или шаблонами вида `{{ .pod_labels.app }}`. Трансформации выполняются после применения `extraLabels`. Если имена лейблов совпадают, у значения из `AddLabels` будет приоритет.

Поле `when` содержит необязательный список условий, объединяемых логическим «И». Слева в условии указывается путь к полю записи лога — строка с ведущей точкой и сегментами через точку (например `.namespace`, `.pod_labels.app`, `.message.level`). Каждый элемент списка может быть одним из следующих типов:

- сравнение значения — `.namespace == "production"`. Поддерживаются операторы `==`, `!=`, `==~` и `!=~`;
- проверка наличия поля — в строке условия указывается только путь без оператора, например, `.pod_labels.team` (поле должно существовать);
- проверка отсутствия поля — `!.pod_labels.legacy`.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: add-env-from-pod
spec:
  ...
  transformations:
    - action: AddLabels
      addLabels:
        when:
          - '.namespace == "production"'
        setLabels:
          .env: prod
          .source_app: "{{ .pod_labels.app }}"

```

Результат преобразования для события в неймспейсе `production` с лейблом пода `app=api`:

```
{... "namespace": "production", "pod_labels": { "app": "api" } "env": "prod", "source_app": "api", ...
}
```

Удаление лейблов

Вы можете использовать трансформацию `DropLabels`, чтобы удалить заданные лейблы из записей логов. У каждого элемента списка `dropLabels.labels` есть поле `label` (путь) и опционально `keepKeys`. Если для элемента задан `keepKeys`, поле `label` должно указывать на **объект**; внутри него удаляются все вложенные ключи, кроме перечисленных в `keepKeys` (имена без ведущей точки). Если `keepKeys` у элемента не задан — путь из `label` удаляется целиком.

`{% alert level="info" %}` Перед применением трансформации `DropLabels` к полю `message` или его вложенным полям необходимо преобразовать запись лога в структурированный объект с помощью трансформации `ParseMessage`. `{% endalert %}`

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: DropLabels
      dropLabels:
        labels:
          - label: .example

```

Пример удаления заданного лейбла из структурированного сообщения

В этом примере показано как вы можете удалить лейбл из структурированного JSON-сообщения. Сначала применяется трансформация `ParseMessage` для парсинга сообщения, после чего применяется `DropLabels` для удаления указанного лейбла.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: DropLabels
      dropLabels:
        labels:
          - label: .message.example

```

Пример изначальной записи в логе:

```

{"msg" : "fetching.module.release", "example": "test"}

```

Результат преобразования:

```

{... "message": {
  "msg" : "fetching.module.release"
}}

```

Отправка логов в Elasticsearch с трансформациями

Ниже приведён пример ресурса `ClusterLogDestination` для отправки логов в Elasticsearch.

В этом примере:

- JSON из поля `message` разбирается и переносится в корень события (`targetLabel: "."`);
- в именах ключей в `pod_labels` точки заменяются на подчёркивания;
- исходное поле `message` удаляется.

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://elasticsearch:9200
    index: logs-%F
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
        targetLabel: "."
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
    - action: DropLabels
      dropLabels:
        labels:
          - label: .message

```

Пример исходной записи: приложение на Java пишет структурированный JSON в stdout, поэтому в поле `message` события попадает JSON-строка. Поле `msg` внутри этой строки содержит текст программного лога.

```

{... "namespace": "prod", "pod_labels": {"app.kubernetes.io/name": "orders", "app": "api"}, "message": "
{\\\"@timestamp\\\":\\\"2025-03-25T10:15:00.123Z\\\",\\\"level\\\":\\\"INFO\\\",\\\"logger\\\":\\\"com.example.OrderService\\\",\\\"msg\\\":\\\"Order id=42
created; user=1001\\\"}\"}

```

Результат после преобразований:

- поля из JSON перенесены в корень события;
- в `pod_labels` точки в именах ключей заменены на подчёркивания;
- поле `message` удалено.

```

{... "namespace": "prod",
  "pod_labels": {
    "app_kubernetes_io/name": "orders",
    "app": "api"
  },
  "@timestamp": "2025-03-25T10:15:00.123Z",
  "level": "INFO",
  "logger": "com.example.OrderService",
  "msg": "Order id=42 created; user=1001"
}

```

Настройка сборки логов с продуктовых неймспейсов, используя опцию `namespace label selector`

```

apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: production-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchLabels:
          environment: production
  destinationRefs:
    - loki-storage

```

Исключение подов и неймспейсов, используя `label`

Существует преднастроенный `label` для исключения определенных подов и неймспейсов: `log-shipper.deckhouse.io/exclude=true`. Он помогает остановить сбор логов с подов и неймспейсов без изменения глобальной конфигурации.


```

---
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    log-shipper.deckhouse.io/exclude: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
spec:
  ...
  template:
    metadata:
      labels:
        log-shipper.deckhouse.io/exclude: "true"

```

Включение буферизации

Настройка буферизации логов необходима для улучшения надежности и производительности системы сбора логов. Буферизация может быть полезна в следующих случаях:

1. Временные перебои с подключением. Если есть временные перебои или нестабильность соединения с системой хранения логов (например, с Elasticsearch), буфер позволяет временно сохранять логи и отправить их, когда соединение восстановится.
2. Сглаживание пиков нагрузки. При внезапных всплесках объема логов буфер позволяет сгладить пиковую нагрузку на систему хранения логов, предотвращая её перегрузку и потенциальную потерю данных.
3. Оптимизация производительности. Буферизация помогает оптимизировать производительность системы сбора логов за счёт накопления логов и отправки их группами, что снижает количество сетевых запросов и улучшает общую пропускную способность.

Пример включения буферизации в оперативной памяти

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    memory:
      maxEvents: 4096
      type: Memory
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Пример включения буферизации на диске

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
      type: Disk
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Пример определения поведения при переполнении буфера

```

apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
      type: Disk
      whenFull: DropNewest
  type: Loki
  loki:
    endpoint: http://loki.loki:3100

```

Более подробное описание параметров доступно в ресурсе ClusterLogDestination.

Модуль loki

⚠ Стадия жизненного цикла модуля: Preview

В Kubernetes системные логи на узлах сохраняются недолго и могут быть утеряны при перезапуске или обновлении. Этот модуль разворачивает в кластере собственное хранилище оперативных логов на базе Grafana Loki.

Возможности:

- системные логи автоматически попадают в Loki без дополнительной настройки;
- доступ к логам реализован через Grafana и веб-интерфейс Deckhouse Kubernetes Platform (console);
- модуль предназначен для хранения логов в течение короткого времени. Для долгосрочного хранения или архивирования рекомендуется использовать внешние системы, поддерживаемые через log-shipper.

Настройки

⚠ Стадия жизненного цикла модуля: Preview

В модуле реализовано 2 алерта.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `loki`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left is a sidebar with the Deckhouse logo and navigation links: 'Обзор', 'Обновления', 'Модули' (selected), 'Глобальные настройки', and 'Kubernetes'. The main area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. It contains a search bar and a table of modules. The table has columns for 'Имя' (Name), 'Состояние' (Status), and 'Этап' (Stage). The 'admission-policy-engine' module is highlighted, and its status is 'ВКЛЮЧЕН' (Enabled). To the right of the table, there is a detailed view of the 'admission-policy-engine' module, showing a toggle switch for 'Модуль включен' (Module enabled) which is currently turned on. Below this, there is a link to 'Документация к настройкам модуля admission-policy-engine'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `loki`:

```
d8 system module enable loki
```

- **С помощью ModuleConfig `loki`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `loki` (создайте его, при необходимости).

Пример манифеста для включения модуля `loki`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  enabled: true

```

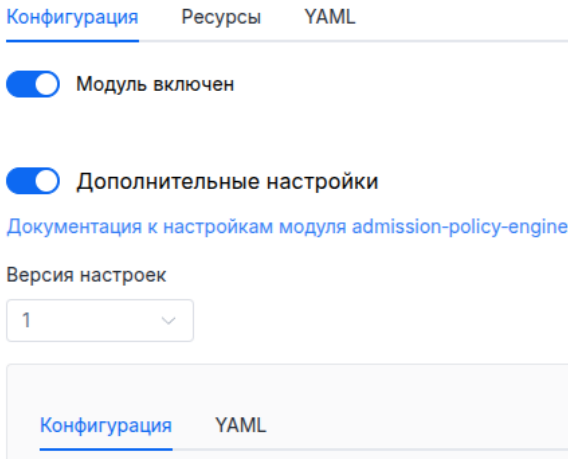
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `loki`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `loki` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `loki`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `loki` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc loki
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `loki`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `loki`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.allowDeleteLogs**
булевый

Включение / выключение эндпоинтов API для удаления логов.

По умолчанию: `false`

- **settings.diskSizeGigabytes**
целочисленный

Начальный размер диска для хранения логов или размер PVC при указании параметра `storageClass`.

Вы должны самостоятельно указать необходимый размер диска для хранения логов за период, указанный в параметре `retentionPeriodHours`.

Занятый объем автоматически мониторится с помощью модуля `extended-monitoring`

По умолчанию: `2`

- **settings.lokiConfig**
объект

Опции Loki, доступные для изменения.

Подробнее о настройке Loki читайте в документации Loki.

i Внимание! Ошибочные настройки могут привести к неработоспособности Loki.

- **settings.lokiConfig.grpcServerMaxConcurrentStreams**
целочисленный

Ограничение на количество одновременных потоков для вызовов gRPC.

По умолчанию: `100`

- **settings.lokiConfig.grpcServerMaxRecvMsgSize**
целочисленный

Ограничение на размер сообщения gRPC, которое может получить сервер, в байтах.

По умолчанию: `104857600`

- **settings.lokiConfig.grpcServerMaxSendMsgSize**
целочисленный

Ограничение на размер сообщения gRPC, которое может отправить сервер, в байтах.

По умолчанию: `4194304`

- **settings.lokiConfig.ingestionBurstSizeMB**
число

Общий лимит всплеска объема принимаемых логов. Например, максимальный размер запроса на запись.

По умолчанию: `6`

- **settings.lokiConfig.ingestionRateMB**
число

Общий лимит скорости приема логов в мегабайтах в секунду.

По умолчанию: `4`

- **settings.lokiConfig.maxChunksPerQuery**
целочисленный

Максимальное количество файлов, которые могут быть скачаны в рамках одного запроса.

По умолчанию: `2000000`

- **settings.lokiConfig.maxEntriesLimitPerQuery**
целочисленный

Максимальное количество записей, возвращаемых для одного запроса.

По умолчанию: `5000`

- **settings.lokiConfig.maxLabelNamesPerSeries**
целочисленный

Максимальное количество лейблов, которые могут быть использованы в одном потоке логов.

По умолчанию: `32`

- **settings.lokiConfig.maxStreamsPerUser**
целочисленный

Максимальное общее количество потоков логов для экземпляра Loki.

Каждый поток логов имеет уникальный набор лейблов. Количество потоков равно количеству уникальных наборов лейблов. Например: `{job="my-app", instance="kube-node-1", namespace="production", pod_name="backend-79dbf6fcb7-v5gs7", app="backend"}`.

Значение 0 отключает ограничение.

По умолчанию: `0`

- **settings.lokiConfig.perStreamRateLimit**

строка or число

Максимальная скорость в байтах в секунду для одного потока. Может быть как числом, так и строкой с единицей измерения (KB, MB, GB).

По умолчанию: `3MB`

Шаблон: `^[0-9]+(KB|MB|GB)?$`

- **settings.lokiConfig.perStreamRateLimitBurst**

строка or число

Максимальный размер всплеска объема логов для одного потока. Может быть как числом, так и строкой с единицей измерения (KB, MB, GB).

По умолчанию: `15MB`

Шаблон: `^[0-9]+(KB|MB|GB)?$`

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.resourcesManagement**

Настройки запросов (requests) и ограничений (limits) использования CPU и памяти подами Loki.

Примеры:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

- **settings.resourcesManagement.mode**

строка

Режим управления ресурсами:

- `Static` — классический, с помощью указания запросов/ограничений (requests/limits). Настраивается в секции параметров `static`;
- `VPA` — с помощью VPA. Настраивается в секции параметров `vpa`.

По умолчанию: `VPA`

Допустимые значения: `VPA`, `Static`

- **settings.resourcesManagement.static**

объект

Настройка управления ресурсами в режиме `Static`.

- **settings.resourcesManagement.static.limits**

объект

Настройка ограничений (limits) использования CPU и памяти.

- **settings.resourcesManagement.static.limits.cpu**

строка от число

Настройка ограничений использования CPU (CPU limits).

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.static.limits.memory**

строка от число

Настройка ограничений использования памяти (memory limits).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.static.requests**

объект

Настройки запросов ресурсов (requests) подов.

- **settings.resourcesManagement.static.requests.cpu**

строка от число

Настройка запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.static.requests.memory**

строка от число

Настройка запроса памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa**

объект

Настройка управления ресурсами в режиме `VPA`.

- **settings.resourcesManagement.vpa.cpu**

объект

Настройки VPA при работе с CPU.

- **settings.resourcesManagement.vpa.cpu.limitRatio**

число

Коэффициент расчета ограничений использования CPU относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования CPU для пода (CPU limits).

Если параметр указан, то VPA, при пересчете запросов CPU (CPU requests) будет пересчитывать ограничения использования CPU (CPU limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.resourcesManagement.vpa.cpu.max**

строка от число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `2`

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.cpu.min**

строка от число

Минимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

По умолчанию: `50m`

Шаблон: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.memory**

объект

Настройки VPA при работе с памятью.

- **settings.resourcesManagement.vpa.memory.limitRatio**

число

Коэффициент расчета ограничений использования памяти относительно запросов (limits/requests).

Применяется для расчета начальных ограничений использования памяти для пода

(memory limits).

Если параметр указан, то VPA, при пересчете запросов памяти (memory requests) будет пересчитывать ограничения использования памяти (memory limits) сохраняя указанное соотношение ограничений к запросам (limits/requests).

- **settings.resourcesManagement.vpa.memory.max**
строка or число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: `2Gi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.memory.min**
строка or число

Минимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

По умолчанию: `256Mi`

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.mode**
строка

Режим работы VPA.

- **Initial** — VPA устанавливает начальные значения requests/limits при создании пода. В дальнейшем значения ресурсов не изменяются автоматически.
- **InPlaceOrRecreate** — VPA пытается изменить ресурсы пода «на месте» (in-place), если это поддерживается кластером. Если in-place обновление невозможно, под будет пересоздан.
- **Auto** — VPA автоматически пересоздаёт поды для применения новых значений ресурсов. Начиная с версии Deckhouse 1.75 данный режим считается устаревшим. Рекомендуется использовать **InPlaceOrRecreate**.

По умолчанию: `InPlaceOrRecreate`

Допустимые значения: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.retentionPeriodHours**
целочисленный

Желаемый минимальный период хранения логов в часах.

Данный параметр не участвует в управлении глубиной хранения логов, так как Loki удаляет старые данные только при заполнении диска.

В случае, если Loki удалит данные, возраст которых еще не достиг указанного периода хранения, вы получите алерт `LokiInsufficientDiskForRetention`. В этой ситуации необходимо увеличить размер диска или уменьшить значение данного параметра.

По умолчанию: `168`

- **settings.storageClass**
строка or булевый

Имя StorageClass для хранения логов Loki (PVC).

Если значение не указано, используется StorageClass существующей PVC Loki. Если PVC нет, будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск будет перезаказан, и все данные удалятся.

Если указать `false`, будет принудительно использоваться `emptyDir`.

Допустимые значения: `false`

Примеры:

```
storageClass: false
```

```
storageClass: default
```

- **settings.storeSystemLogs**
булевый

Сохранять логи из пространства имен `d8-*` в loki.

Для работы необходим включенный модуль log-shipper.

По умолчанию: `true`

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Пример конфигурации

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  settings:
    storageClass: ceph-csi-rbd
    diskSizeGigabytes: 10
    retentionPeriodHours: 48
  enabled: true
  version: 1
```

FAQ



Стадия жизненного цикла модуля: Preview

Почему срабатывает алерт `LokiDiscardedSamplesWarning`?

Алерт `LokiDiscardedSamplesWarning` срабатывает, когда Loki отказывается принимать логи во время приёма данных, потому что они не проходят валидацию или превышают установленные лимиты.

Иными словами, `log-shipper` пытается отправить данные, которые Loki не принимает.

Возможные причины

Вероятнее всего превышен лимит на размер или количество стримов.

Loki ограничивает:

- количество уникальных лейблов в стриме;
- длину лейблов и их значений;
- количество стримов от одного клиента;
- размер пакета данных (batch) при приёме.

Подробное описание каждого ограничения можно найти в официальной документации Loki.

Устранение проблемы

1. Уточните причину в описании алерта.

```
Samples are being discarded because of "{{ $labels.reason }}"...
```

Поле `{{ $labels.reason }}` указывает на сработавший лимит и причину, по которой Loki не принимает логи.

2. Отредактируйте соответствующий лимит в конфигурации модуля `loki`.

Описание доступных для настройки полей можно найти в документации модуля.

Например, если в качестве причины указан `rate_limited`, необходимо увеличить `ingestionRateMB` или `ingestionBurstSizeMB`, если возможны временные повышения объёма логов. В качестве альтернативы можно ограничить количество отсылаемых логов на стороне приложения.

Примеры



Стадия жизненного цикла модуля: Preview

Чтение логов из всех подов из указанного namespace и направление их в Loki


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  settings:
    storageClass: ceph-csi-rbd
    diskSizeGigabytes: 30
    retentionPeriodHours: 168
  enabled: true
  version: 1
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: development-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: "kubernetes.io/metadata.name"
            operator: In
            values: [development]
  destinationRefs:
    - d8-loki

```

Больше примеров в описании модуля log-shipper.

Модуль metallb

Модуль реализует механизм `LoadBalancer` для сервисов в кластерах bare metal.

Поддерживает следующие режимы работы:

- **Режим Layer 2** — реализует улучшенный (относительно стандартного режима L2 в MetallB) механизм балансировки в кластерах bare metal, который позволяет использовать несколько «публичных» адресов для сервисов кластера.
- **Режим BGP** — полностью основан на решении MetallB.

Режим Layer 2

В режиме Layer 2 один или несколько узлов берут на себя ответственность за предоставление сервиса в «публичной» сети. С точки зрения сети это выглядит, как будто каждый из узлов имеет несколько IP-адресов, назначенных его сетевому интерфейсу. Технически это реализовано следующим образом: модуль отвечает на ARP-запросы для IPv4-сервисов и NDP-запросы — для IPv6. Основное преимущество режима layer 2 — его универсальность: он работает в любой сети Ethernet, не требуя специального оборудования.

Преимущества модуля перед классическим MetallB

MetallB в режиме L2 позволяет заказать Service с типом `LoadBalancer`, работа которого основана на том, что в пиринговой сети узлы-балансировщики имитируют ARP-ответы от «публичного» IP. Этот режим имеет существенное ограничение — одновременно лишь один балансировочный узел обрабатывает весь входящий трафик данного сервиса. Соответственно:

- Узел, выбранный в качестве лидера для «публичного» IP становится «узким местом» без возможности горизонтального масштабирования.
- В случае выхода узла-балансировщика из строя, все текущие соединения будут разорваны в процессе переключения на новый узел, который будет назначен лидером.

Данный модуль помогает обойти эти ограничения. Он предоставляет новый ресурс `MetalLoadBalancerClass`, который позволяет с помощью `nodeSelector` связать группу узлов и пул IP-адресов. После чего можно создать стандартный ресурс Service с типом `LoadBalancer` и в нем указать имя обслуживающего `MetalLoadBalancerClass`'а, а с помощью аннотации указать необходимое количество IP-адресов для L2-анонсирования.

Таким образом:

- Приложение получит не один, а несколько «публичных» IP. Эти IP потребуются прописать в качестве A-записей для публичного домена приложения. Для последующего горизонтального масштабирования потребуются добавить дополнительные узлы-балансировщики. Соответствующие Service будут созданы автоматически, потребуется лишь добавить их в список A-записей прикладного домена.
- При выходе из строя одного из узлов-балансировщиков, лишь часть трафика будет подвержена обрыву для переключения на здоровый узел.

Режим BGP

Metallb в режиме BGP предоставляет эффективный и масштабируемый способ предоставления Service'ов типа `LoadBalancer` в кластерах Kubernetes, работающих на «голом железе». Благодаря использованию стандартизированного протокола BGP, metallb легко интегрируется в существующую сетевую инфраструктуру и обеспечивает высокую доступность Service'ов.

Принцип работы metallb в режиме BGP

В режиме BGP metallb устанавливает BGP-сессии с маршрутизаторами (или с Top-of-Rack коммутаторами) и анонсирует им IP-адреса сервисов типа `LoadBalancer`. Это достигается следующим образом:

Конфигурация.

- Metallb конфигурируется с указанием пула IP-адресов, которые он может назначать Service'ам.
- Определяются параметры BGP-сессий: AS (Autonomous System) number кластера Kubernetes, IP-адреса маршрутизаторов (пиров), AS number пиров, пароли для аутентификации (опционально).
- Для каждого пула IP-адресов можно задать специфические параметры анонсирования, например, `community strings`.

Установка BGP-сессий.

- На каждом узле кластера Kubernetes, где запущен metallb, компонент `speaker` устанавливает BGP-сессии с указанными маршрутизаторами.
- Происходит обмен маршрутной информацией между metallb и маршрутизаторами.

Назначение IP-адресов Service'ам.

- Когда создается Service типа `LoadBalancer`, metallb выбирает свободный IP-адрес из сконфигурированного пула и назначает его Service'у.
- Компонент `controller` отслеживает изменения в Service'ах и управляет назначениями IP-адресов.

Анонсирование IP-адресов.

- После назначения IP-адреса `speaker` на узле, выбранном лидером для данного Service'а, начинает анонсировать этот IP-адрес через установленные BGP-сессии.
- Маршрутизаторы получают этот анонс и обновляют свои таблицы маршрутизации, направляя трафик к этому IP-адресу на узел, анонсирующий его.

Распределение трафика.

- Маршрутизаторы используют протоколы ECMP (Equal-Cost Multi-Path) или другие алгоритмы балансировки для распределения трафика между узлами, анонсирующими один и тот же IP-адрес Service'а.
- Внутри кластера Kubernetes трафик, поступивший на узел, перенаправляется на поды Service'а с помощью механизмов используемого CNI (iptables/IPVS, eBPF-программы и т.д.).

Преимущества использования BGP

- **Стандартизированный протокол.** BGP — широко используемый и хорошо зарекомендовавший себя протокол маршрутизации.
- **Гибкость и масштабируемость.** BGP позволяет интегрировать кластер Kubernetes в существующую сетевую инфраструктуру и масштабировать сервисы.
- **Распределение нагрузки.** Использование ECMP позволяет эффективно распределять нагрузку между узлами кластера.
- **Отказоустойчивость.** При выходе из строя узла, анонсирующего IP-адрес, маршрутизаторы автоматически перенаправляют трафик на другие узлы, анонсирующие этот же IP.

Недостатки использования BGP

- **Сложность настройки.** Настройка BGP может быть сложнее, чем настройка анонсов ARP/GARP (в режиме Layer 2).
- **Требования к сетевому оборудованию.** Маршрутизаторы должны поддерживать BGP и ECMP.

Настройки

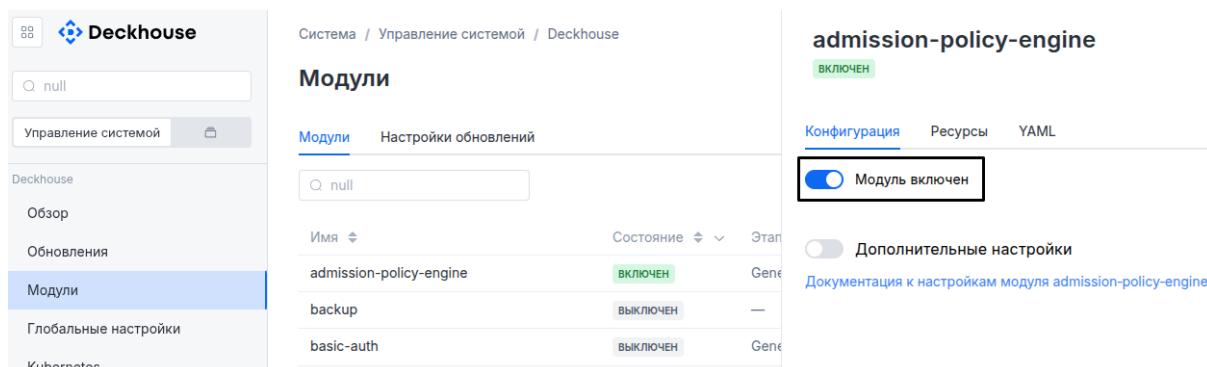
В модуле реализовано 7 алертов.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `metallb`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable metallb` для включения модуля, или `d8 system module disable metallb` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `metallb`:

```
d8 system module enable metallb
```

- **С помощью ModuleConfig `metallb`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `metallb` (создайте его, при необходимости).

Пример манифеста для включения модуля `metallb`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `metallb`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `metallb` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `metallb`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `metallb` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc metallb
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `metallb`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `metallb`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Удалите все элементы `addressPool` с протоколом `layer2`.

Параметры

Версия схемы: 2

Пример:

```

bgpPeers:
- peer-address: 192.168.1.1
  peer-asn: 1111
  my-asn: 2222
  source-address: 192.168.1.2
  hold-time: 10s
  node-selector:
    matchLabels:
      node: test
addressPools:
- name: my-pool-bgp
  protocol: bgp
  addresses:
  - 192.168.100.1-192.168.100.10
  - 192.168.101.0/24
  bgp-advertisements:
  - aggregation-length: 32
    localpref: 100
    communities:
    - no-advertise
bgpCommunities:
  no-advertise: 65535:65282
speaker:
  nodeSelector:
    mylabel: speaker

```

- **settings**

объект

- **settings.addressPools**

массив объектов

Обязательный параметр

Список диапазонов IP, которые будут выдаваться сервисам.

Формат — массив, как у оригинального MetalLB.

По умолчанию: `[]`

- **settings.addressPools.addresses**

массив строк

Обязательный параметр

Список диапазонов, где диапазон — это либо подсеть с маской, либо диапазон значений с “-” в качестве разделителя.

- **settings.addressPools.auto-assign**

булевый

Флаг auto-assign используется для предотвращения автоматического выделения IP из пула.

По умолчанию: `true`

- **settings.addressPools.avoid-buggy-ips**

булевый

Запрещает использование адресов из пула, заканчивающихся на `.0` и `.255`.

По умолчанию: `false`

■ **settings.addressPools.bgp-advertisements**

массив объектов

Определяет BGP advertisements.

■ **settings.addressPools.bgp-advertisements.aggregation-length** целочисленный

Опция aggregation-length позволяет «свернуть» префикс в более крупный.

Работает для адресов IPv4.

По умолчанию: `32`

Допустимые значения: `1 <= X`

■ **settings.addressPools.bgp-advertisements.communities**

массив строк

BGP communities, связанные с анонсом. Каждый элемент — это ключ из параметра bgpCommunities.

Пример:

```
communities:
- no-advertise
```

■ **settings.addressPools.bgp-advertisements.localpref**

целочисленный

Атрибут BGP LOCAL_PREF используется алгоритмом BGP best path.

Путь с более высоким localpref предпочтительнее пути с более низким localpref.

■ **settings.addressPools.name**

строка

Обязательный параметр

Имя пула. Должно соответствовать RFC 1123: разделенные точками части в нижнем регистре, состоящие из букв и '-'. Каждое имя должно начинаться и заканчиваться буквой алфавита.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ **settings.addressPools.protocol**

строка

Обязательный параметр

Технология публикации сервисов, которую должен реализовать speaker.

Допустимые значения: `bgp`, `layer2`

○ **settings.bgpCommunities**

объект

Доступно в редакциях: EE, CSE Pro (1.73)

Список BGP communities.

Пример:

```
bgpCommunities:
no-advertise: 65535:65282
```

○ **settings.bgpPeers**

массив объектов

Доступно в редакциях: EE, CSE Pro (1.73)

Список внешних BGP-роутеров, которые будут работать с этим модулем.

Формат — массив, как у оригинального MetalLB.

По умолчанию: `[]`

■ **settings.bgpPeers.hold-time**

строка или число

Таймаут, после которого партнерский BGP-пир считается утерянным. Это значение делится на 3 и получается интервал для keep-alive.

Рекомендуемое значение — `3s` (keep-alive раз в секунду), протокол BGP не поддерживает меньшие значения. По умолчанию — `90s` (keep-alive раз в 30 секунд).

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

- **settings.bgpPeers.my-asn**

целочисленный

Обязательный параметр

Номер автономной системы для кластера.

Допустимые значения: `0 <= X <= 4294967295`

- **settings.bgpPeers.node-selector**

объект

Дополнительный псевдоселектор, который реализуется приложением speaker'a. Говорит о том, с каких узлов стоит устанавливать связь с внешними BGP-роутерами. Не путать с `speaker.nodeSelector` и `nodeSelector`.

Опциональный параметр.

Формат — `matchLabels` или `matchExpressions`.

- **settings.bgpPeers.node-selector.matchExpressions**

массив объектов

- **settings.bgpPeers.node-selector.matchExpressions.key**

строка

- **settings.bgpPeers.node-selector.matchExpressions.operator**

строка

- **settings.bgpPeers.node-selector.matchExpressions.values**

массив строк

- **settings.bgpPeers.node-selector.matchLabels**

объект

- **settings.bgpPeers.password**

строка

Пароль аутентификации для BGP-маршрутизаторов, использующих TCP MD5-аутентифицированные сессии.

- **settings.bgpPeers.peer-address**

строка

Обязательный параметр

IP внешнего BGP-роутера.

Шаблон: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

- **settings.bgpPeers.peer-asn**

целочисленный

Обязательный параметр

Номер автономной системы на внешнем BGP-роутере.

Допустимые значения: `0 <= X <= 4294967295`

- **settings.bgpPeers.peer-port**

целочисленный

Порт, к которому устанавливать сессию.

По умолчанию: `179`

Допустимые значения: `0 <= X <= 16384`

- **settings.bgpPeers.router-id**

строка

ID BGP-роутера для анонса.

- **settings.bgpPeers.source-address**

строка

IP-адрес для исходящих соединений.

Шаблон: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$`

- **settings.loadBalancerClass**

строка

Необязательное поле, описывающее класс LoadBalancer. Поле LoadBalancerClass следует использовать в режиме L2 LoadBalancer для указания MetalLoadBalancerClass, задающего параметры балансировщика для Service.

- **settings.nodeSelector**

объект

Селектор узлов, на которых будет работать главный контроллер. Структура, аналогичная параметру `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.speaker**

объект

Настройки для компонента `speaker`, который реализует протокол публикации IP-адресов LoadBalancer'a

для внешней инфраструктуры.

- **settings.speaker.nodeSelector**
объект

Обязательный параметр

Селектор узлов, на которых будет работать DaemonSet speaker. Структура, аналогичная параметру `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.speaker.tolerations**
массив объектов

Структура для DaemonSet speaker'a, аналогичная параметру `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.speaker.tolerations.effect**
строка
- **settings.speaker.tolerations.key**
строка
- **settings.speaker.tolerations.operator**
строка
- **settings.speaker.tolerations.tolerationSeconds**
целочисленный
- **settings.speaker.tolerations.value**
строка

- **settings.tolerations**
массив объектов

Структура, аналогичная параметру `spec.tolerations` пода Kubernetes для главного контроллера.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Custom resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

MetalLoadBalancerClass

Short names: mlbc

Scope: Cluster

Version: v1alpha1

Описывает ресурс MetalLoadBalancerClass для кластера. Данный класс можно использовать для настройки сервиса с типом `LoadBalancer`, указав в поле соответствующий `spec.loadBalancerClass`. Если в кластере присутствует MetalLoadBalancerClass с флагом `spec.isDefault`, установленным в `true`, явное указание может быть не обязательным.

- **apiVersion**
строка
- **kind**
строка
- **metadata**
объект
- **spec**
объект
 - **spec.addressPool**
массив строк

Массив диапазонов публичных IP-адресов. Каждый диапазон может быть указан с помощью CIDR или прямого указания.

Примеры:

```
addressPool: 10.42.0.0/24
```

```
addressPool: 10.33.5.100-10.33.5.106
```

- **spec.isDefault**

булевый

Выбран ли MetalLoadBalancerClass для обслуживания всех сервисов с типом `LoadBalancer` по умолчанию, без указания `spec.loadBalancerClass`.

Пример:

```
isDefault: true
```

- **spec.l2**
объект

Доступно в редакциях: SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

- **spec.l2.interfaces**
массив строк

Доступно в редакциях: SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Список сетевых интерфейсов, с которых будет производиться анонс присвоенных IP адресов. Если поле не заполнено или задан пустой массив, то будет происходить анонс со всех интерфейсов узла. Настоятельно рекомендуется настраивать этот параметр и не допускать анонса адресов со всех интерфейсов узла.

Пример:

```
interfaces:
- eth1
- eth2
```

- **spec.nodeSelector**
объект

Позволяет ограничить список узлов, с которых будет происходить анонс.

Задается в виде набора лейблов для фильтрации узлов.

- **spec.type**
строка

Выбор режима работы балансировщика нагрузки.

Пример:

```
type: L2
```

- **status**
объект

The metallb module: примеры

Metallb можно использовать в статических кластерах (bare metal), когда нет возможности воспользоваться балансировщиком от облачного провайдера. Metallb может работать в режимах L2 LoadBalancer или BGP LoadBalancer.

Пример использования metallb в режиме L2 LoadBalancer

Включите модуль:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

Подготовьте приложение, которое хотите опубликовать:

```
d8 k create deploy nginx --image=nginx
```

Создайте ресурс MetalLoadBalancerClass:


```

apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: "" # селектор узлов-балансировщиков
  type: L2

```

Создайте ресурс Service с аннотацией и именем MetalLoadBalancerClass:

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # имя MetalLoadBalancerClass
  ports:
    - port: 8000
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx

```

В результате, созданному сервису с типом `LoadBalancer` будут присвоены адреса в заданном количестве:

```
d8 k get svc
```

Пример вывода:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx-deployment	LoadBalancer	10.222.130.11	192.168.2.100,192.168.2.101,192.168.2.102	80:30544/TCP	11s

i При создании сервиса также можно указать определенные IP-адреса из пула, которые будут ему присвоены.

Полученные EXTERNAL-IP можно прописывать в качестве A-записей для прикладного домена:

```

$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.100:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.101:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.102:8000
200

```

Пример использования metallb в режиме BGP LoadBalancer

Включите модуль и настройте все необходимые параметры*:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  settings:
    addressPools:
    - addresses:
      - 192.168.219.100-192.168.219.200
      name: mypool
      protocol: bgp
    bgpPeers:
    - hold-time: 3s
      my-asn: 64600
      peer-address: 172.18.18.10
      peer-asn: 64601
    speaker:
      nodeSelector:
        node-role.deckhouse.io/metallb: ""
  version: 2

```

* — в будущих версиях настройки режима BGP будут задаваться через ресурс MetalLoadBalancerClass.

Настройте BGP-пиринг на сетевом оборудовании.

Дополнительные примеры настроек для Service

Создание сервиса с присвоением ему определенных IP-адресов из пула

i Для указания адресов, которые должны быть присвоены сервису, используйте аннотацию `network.deckhouse.io/load-balancer-ips`. Если желаемых адресов больше одного, то также должна присутствовать аннотация `network.deckhouse.io/l2-load-balancer-external-ips-count`, в которой необходимо указать количество выделяемых адресов из пула (оно не должно быть меньше количества адресов, перечисленных в `network.deckhouse.io/load-balancer-ips`).

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # Количество адресов, которые будут выделены из пула, объявленного в MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # Список адресов из пула, объявленного в MetalLoadBalancerClass, которые будут выделены сервису.
    network.deckhouse.io/load-balancer-ips: "192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # имя MetalLoadBalancerClass
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx

```

В результате, созданному сервису с типом `LoadBalancer` будут присвоены указанные адреса:

```
d8 k get svc
```

Пример вывода:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx-deployment	LoadBalancer	10.222.130.11	192.168.2.102,192.168.2.103,192.168.2.104	80:30544/TCP	11s

Создание сервиса с одним принудительно выбранным адресом

Для создания Service с принудительно выбранным адресом необходимо добавить аннотацию `network.deckhouse.io/load-balancer-ips`:

```

apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    network.deckhouse.io/load-balancer-ips: 192.168.217.217
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer

```

Создание сервисов с общими IP адресами

Для создания Services с общими IP адресами необходимо добавить к ним аннотацию `network.deckhouse.io/load-balancer-shared-ip-key`. Общий ключ является значением аннотации. Services могут совместно использовать IP-адрес при следующих условиях:

- Они имеют одинаковый ключ совместного использования.
- Они запрашивают использование разных портов (например, tcp/80 для одного и tcp/443 для другого).
- Они используют Cluster external traffic policy или указывают на один и тот же набор pods (т.е. селекторы pods идентичны).

Если эти условия выполняются, MetalLB может разместить два сервиса на одном IP-адресе, но **не обязан этого делать**! Если вы хотите гарантировать, что они будут совместно использовать определенный адрес, используйте аннотацию `network.deckhouse.io/load-balancer-ips`, описанную выше.

Ниже приведен пример конфигурации двух сервисов, которые совместно используют один IP-адрес:

```

apiVersion: v1
kind: Service
metadata:
  name: dns-service-tcp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  ports:
    - name: dnstcp
      protocol: TCP
      port: 53
      targetPort: 53
  selector:
    app: dns
---
apiVersion: v1
kind: Service
metadata:
  name: dns-service-udp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  ports:
    - name: dnsudp
      protocol: UDP
      port: 53
      targetPort: 53
  selector:
    app: dns

```

Создание сервиса и назначение ему IPAddressPools при использовании metallb в режиме BGP LoadBalancer

Создание Service и назначение ему IPAddressPools возможно в режиме BGP LoadBalancer через аннотацию `metallb.universe.tf`. Для режима L2 LoadBalancer необходимо использовать настройки MetalLoadBalancerClass (см. выше).

```

apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    metallb.universe.tf/address-pool: production-public-ips
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer

```

Автоматическая настройка системы мониторинга для сбора метрик с пользовательских приложений

Для мониторинга пользовательских приложений, запущенных в кластере, требуется дополнительная настройка. Этот модуль упрощает процесс настройки, сводя её к указанию для нужного приложения определённого лейбла.

Чтобы организовать сбор метрик с приложений модулем `monitoring-custom`, необходимо:

- Поставить лейбл `prometheus.deckhouse.io/custom-target` на Service или под. Значение лейбла определит имя в списке `target'ов` Prometheus.
 - В качестве значения лейбла `prometheus.deckhouse.io/custom-target` рекомендуется использовать название приложения (маленькими буквами, разделитель `-`), которое позволяет его уникально идентифицировать в кластере.
- Порту, с которого нужно собирать метрики, указать имя `http-metrics` и `https-metrics` для подключения по HTTP или HTTPS соответственно.

Если это невозможно (например, порт уже определен и назван другим именем), необходимо воспользоваться аннотациями: `prometheus.deckhouse.io/port`: номер порта — для указания порта и `prometheus.deckhouse.io/tls`: "true" — если сбор метрик будет проходить по HTTPS.

i При указании аннотации на Service в качестве значения порта необходимо использовать `targetPort`. То есть тот порт, что открыт и слушается приложением, а не порт Service'a.

- Пример 1:

```

ports:
  - name: https-metrics
    containerPort: 443

```

- Пример 2:

```

annotations:
  prometheus.deckhouse.io/port: "443"
  prometheus.deckhouse.io/tls: "true" # Если метрики отдаются по HTTP, эту аннотацию
указывать не нужно.

```

- При использовании service mesh Istio в режиме STRICT mTLS указать для сбора метрик следующую аннотацию у Service или Pod: `prometheus.deckhouse.io/istio-mtls`: "true". Важно, что метрики приложения должны экспортироваться по протоколу HTTP без TLS.
 - (Необязательно) Укажите дополнительные аннотации для более тонкой настройки:
 - `prometheus.deckhouse.io/path` — путь для сбора метрик (по умолчанию: `/metrics`).
 - `prometheus.deckhouse.io/query-param-$name` — GET-параметры, будут преобразованы в map вида `$name=$value` (по умолчанию: `""`):
 - возможно указать несколько таких аннотаций.
- Например, `prometheus.deckhouse.io/query-param-foo=bar` и `prometheus.deckhouse.io/query-param-bar=zxc` будут преобразованы в query: `http://...?foo=bar&bar=zxc`.
- `prometheus.deckhouse.io/allow-unready-pod` — разрешает сбор метрик с подов в любом состоянии (по умолчанию метрики собираются только с подов в состоянии Ready). Эта опция полезна в редких случаях. Например, если ваше приложение запускается очень долго (при старте загружаются данные в базу или заполняются кеша), но в процессе запуска уже отдаются полезные метрики, которые помогают следить за запуском приложения.
 - `prometheus.deckhouse.io/sample-limit` — сколько семплов разрешено собирать с пода (по умолчанию 5000). Значение по умолчанию защищает от ситуации, когда приложение внезапно начинает отдавать

слишком большое количество метрик, что может нарушить работу всего мониторинга. Аннотация должна быть размещена на том же ресурсе, на котором расположен лейбл `prometheus.deckhouse.io/custom-target`.

Пример: Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    prometheus.deckhouse.io/custom-target: my-app
  annotations:
    prometheus.deckhouse.io/port: "8061" # По умолчанию будет использоваться порт
сервиса с именем http-metrics или https-metrics.
    prometheus.deckhouse.io/path: "/my_app/metrics" # По умолчанию /metrics.
    prometheus.deckhouse.io/query-param-format: "prometheus" # По умолчанию ''.
    prometheus.deckhouse.io/allow-unready-pod: "true" # По умолчанию поды НЕ в Ready
игнорируются.
    prometheus.deckhouse.io/sample-limit: "5000" # По умолчанию принимается не больше 5000
метрик от одного пода.
spec:
  ports:
    - name: my-app
      port: 8060
    - name: http-metrics
      port: 8061
      targetPort: 8061
  selector:
    app: my-app
```

Пример: Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        prometheus.deckhouse.io/custom-target: my-app
    annotations:
      prometheus.deckhouse.io/sample-limit: "5000" # По умолчанию принимается не больше 5000 метрик
от одного пода.
    spec:
      containers:
        - name: my-app
          image: my-app:1.7.9
          ports:
            - name: https-metrics
              containerPort: 443
```

Настройки

Модуль работает, если включен модуль `prometheus`, и не имеет параметров для настройки.

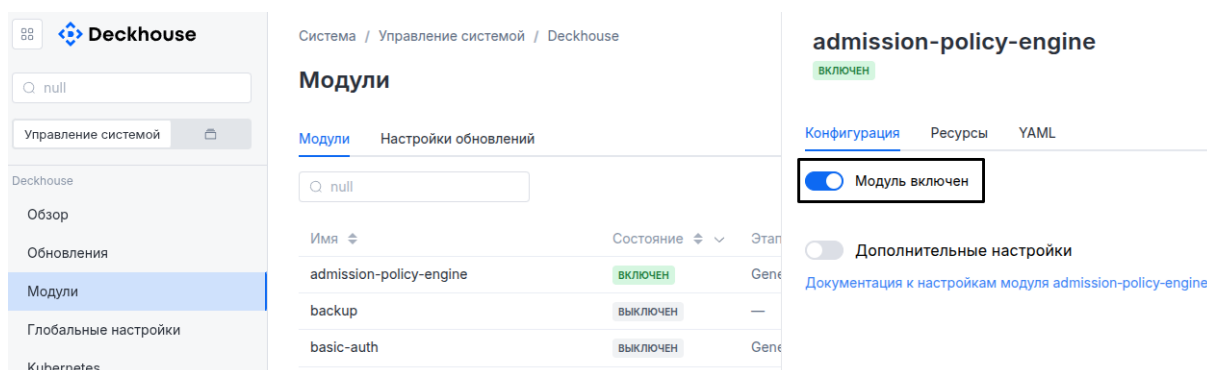
В модуле реализовано 5 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-custom`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `monitoring-custom`:

```
d8 system module enable monitoring-custom
```

- **С помощью ModuleConfig `monitoring-custom`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `monitoring-custom` (создайте его, при необходимости).

Пример манифеста для включения модуля `monitoring-custom`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-custom
spec:
  enabled: true
```

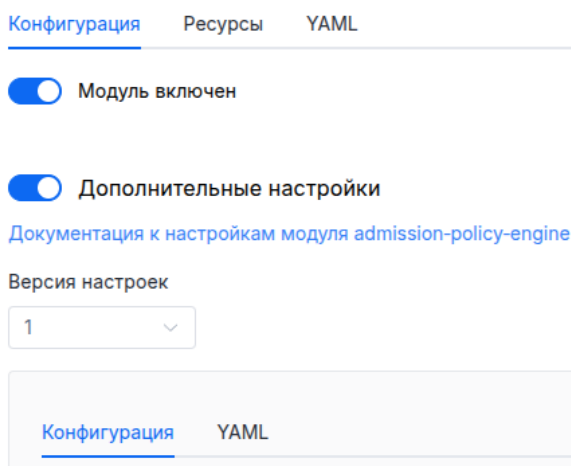
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-custom`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `monitoring-custom` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-custom`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `monitoring-custom` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc monitoring-custom
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `monitoring-custom`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `monitoring-custom`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-custom
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

Модуль monitoring-deckhouse

Модуль `monitoring-deckhouse` обеспечивает мониторинг, оповещения и наблюдаемость самой платформы Deckhouse Kubernetes Platform. Он отслеживает состояние, производительность и правильную работу основных компонентов DKP для обеспечения стабильности и надежности платформы.

Этот модуль — важный компонент обеспечения наблюдаемости, который работает совместно с модулем `prometheus` для предоставления информации о рабочем состоянии DKP.

Модуль разворачивает ресурсы мониторинга, которые:

- **Собирают метрики DKP** — модуль скрейпит метрики из пода deckhouse с использованием PodMonitor ресурсов, включая:
 - Сами метрики на порту 4222 через эндпоинт `/metrics`.
 - Пользовательские метрики, генерируемые хуками, через `/metrics/hooks`.
 - Метрики выполнения модулей, хуков и системного здоровья.
- **Определяют правила оповещений** — модуль предоставляет комплексные правила оповещений Prometheus, организованные по категориям:
 - **Доступность DKP** — мониторинг здоровья подов, готовности и времени работы.
 - **Неисправности DKP** — обнаружение чрезмерных перезапусков, проблем доступа к container registry, зависших процессов.
 - **Управление релизами** — отслеживание подписок на каналы релизов, ожидающих обновлений и ручных подтверждений.
 - **Управление модулями** — мониторинг состояния модулей, ошибок валидации и устаревших конфигураций.
 - **Проверки CNI** — обнаружение нескольких конфигураций CNI и их некорректности.
 - **Требования ОС** — идентификация узлов с устаревшими версиями операционной системы.
- **Предоставляет дашборды Grafana** — модуль включает предварительно созданные дашборды Grafana для визуализации:
 - Метрик производительности DKP.
 - Статистики работы модулей.
 - Времени выполнения хуков и использования ресурсов.
 - Обработки очередей и статуса конвергенции.

Сбор метрик

Модуль настраивает PodMonitor, который скрейпит два эндпоинта из пода Deckhouse:

1. **Метрики DKP** (`/metrics`) — Основные операционные метрики DKP, в том числе:
 - `deckhouse live ticks` — индикатор здоровья, увеличивающийся каждые 10 секунд.
 - `deckhouse registry errors` — проблемы подключения к container registry.
 - `deckhouse module hook run seconds` — время выполнения хуков модулей.
 - `deckhouse_tasks queue action_duration_seconds` — время обработки очередей задач.
2. **Метрики хуков** (`/metrics/hooks`) — пользовательские метрики, генерируемые хуками Deckhouse с `honorLabels: true` для сохранения лейблов, специфичных для хуков.

Интеграция с модулем Observability

При включении модуля `observability` этот модуль автоматически создает:

- Ресурсы ClusterObservabilityMetricsRulesGroup для правил Prometheus.
- Ресурсы ClusterObservabilityDashboard для дашбордов Grafana.

Это обеспечивает централизованное управление и поддержку мультиарендности для ресурсов мониторинга.

Требования

- Модуль `prometheus` должен быть включен (автоматическая зависимость).
- Модуль `operator-prometheus` должен быть включен для поддержки PodMonitor.

Настройки

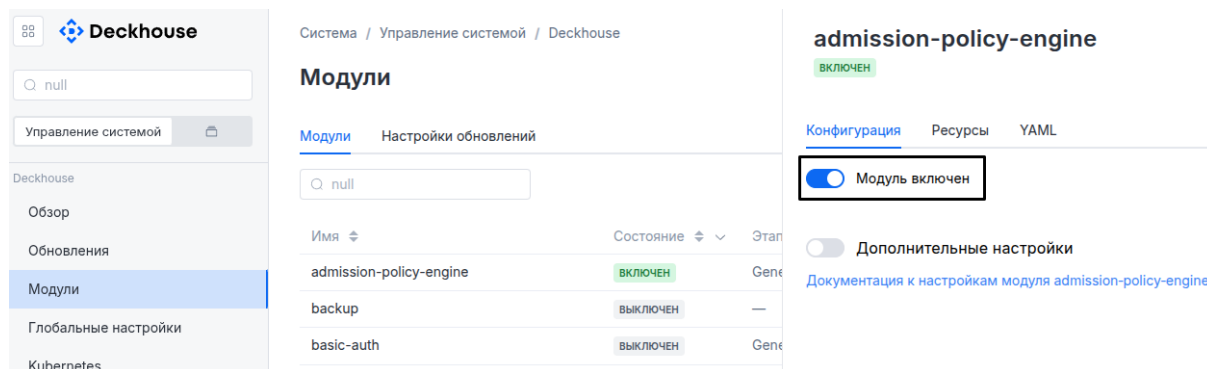
В модуле реализовано 51 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-deckhouse`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `monitoring-deckhouse`:

```
d8 system module enable monitoring-deckhouse
```

- **С помощью ModuleConfig monitoring-deckhouse.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig monitoring-deckhouse` (создайте его, при необходимости).

Пример манифеста для включения модуля `monitoring-deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-deckhouse
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-deckhouse`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `monitoring-deckhouse` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-deckhouse`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `monitoring-deckhouse` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc monitoring-deckhouse
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `monitoring-deckhouse`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `monitoring-deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-deckhouse
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

Модуль не имеет настроек.

Модуль monitoring-kubernetes

Модуль `monitoring-kubernetes` обеспечивает прозрачный и своевременный контроль состояния всех узлов кластера и ключевых инфраструктурных компонентов.

Возможности модуля:

- предоставляет возможность планировать ресурсы инфраструктуры (Capacity planning);
- отслеживает версию container runtime (docker, containerd) на каждом узле и проверяет её на соответствие разрешенным версиям;
- контролирует работоспособность самой подсистемы мониторинга кластера (Dead man's switch);
- снимает метрики о доступности файловых дескрипторов, сокетов, свободного места и inode на каждом узле;
- следит за корректной работой ключевых компонентов мониторинга: kube-state-metrics, node-exporter, kube-dns;
- проверяет состояние всех узлов (NotReady, drain, cordon) и своевременно сигнализирует о неполадках;
- следит за синхронизацией времени и уведомляет об отклонениях;
- выявляет случаи продолжительного превышения CPU steal (когда узел не получает нужного времени процессора);
- контролирует состояние таблицы Conntrack на узлах;
- показывает поды с некорректными статусами — например, если kubelet не справился со своей работой;
- позволяет экспортировать метрики во внешние системы мониторинга для единой точки контроля.

Настройки

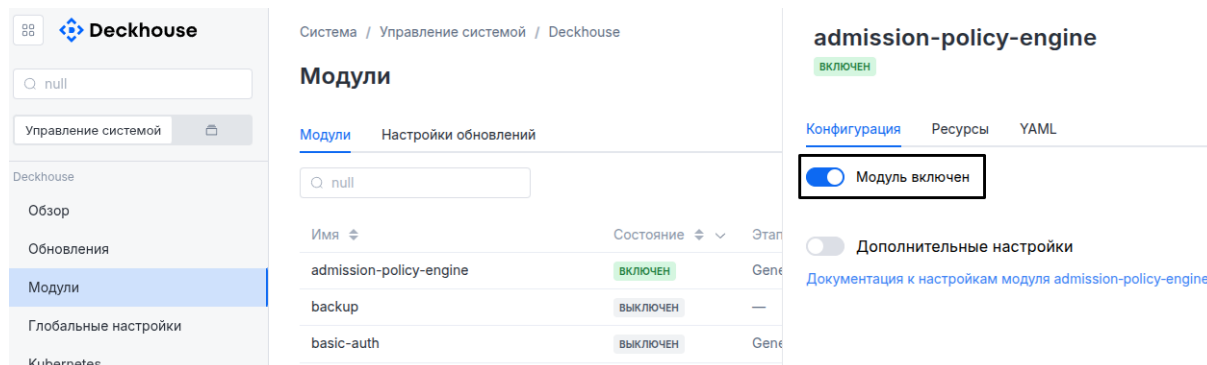
В модуле реализовано 47 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `monitoring-kubernetes`:

```
d8 system module enable monitoring-kubernetes
```

- **С помощью ModuleConfig `monitoring-kubernetes`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `monitoring-kubernetes` (создайте его, при необходимости).

Пример манифеста для включения модуля `monitoring-kubernetes`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `monitoring-kubernetes` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `monitoring-kubernetes` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc monitoring-kubernetes
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `monitoring-kubernetes`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `monitoring-kubernetes`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Если для поля `.ebpfExporterEnabled` установлено значение `true`, добавьте поле `.oomKillsExporterEnabled=true`. После этого удалите поле `.ebpfExporterEnabled` из объекта.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.highAvailability**
булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим

отказоустойчивости.

Примеры:

```
highAvailability: true
```

```
highAvailability: false
```

◦ **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
nodeSelector:
  disktype: ssd
```

◦ **settings.oomKillsExporterEnabled**

булевый

Флаг отключения oom-kills-exporter для случаев, когда его работа создает проблемы в нормальной работе узлов.

По умолчанию: `true`

Пример:

```
oomKillsExporterEnabled: false
```

◦ **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

▪ **settings.tolerations.effect**

строка

▪ **settings.tolerations.key**

строка

▪ **settings.tolerations.operator**

строка

▪ **settings.tolerations.tolerationSeconds**

целочисленный

▪ **settings.tolerations.value**

строка

◦ **settings.vpa**

объект

Настройки VPA для подов `kube-state-metrics`.

Пример:

```
vpa:
  kubeStateMetricsMaxCPU: 130m
  kubeStateMetricsMaxMemory: 115Mi
```

▪ **settings.vpa.kubeStateMetricsMaxCPU**

строка or число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests).

Шаблон: `^[0-9]+m?$`

Примеры:

```
kubeStateMetricsMaxCPU: 3
```

```
kubeStateMetricsMaxCPU: 1000m
```

▪ **settings.vpa.kubeStateMetricsMaxMemory**

строка or число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
kubeStateMetricsMaxMemory: 3Mi
```

Мониторинг control plane

Мониторинг control plane осуществляется с помощью модуля `monitoring-kubernetes-control-plane`, который организует безопасный сбор метрик и предоставляет базовый набор правил мониторинга следующих компонентов кластера:

- kube-apiserver;
- kube-controller-manager;
- kube-scheduler;
- kube-etcd.

Настройки

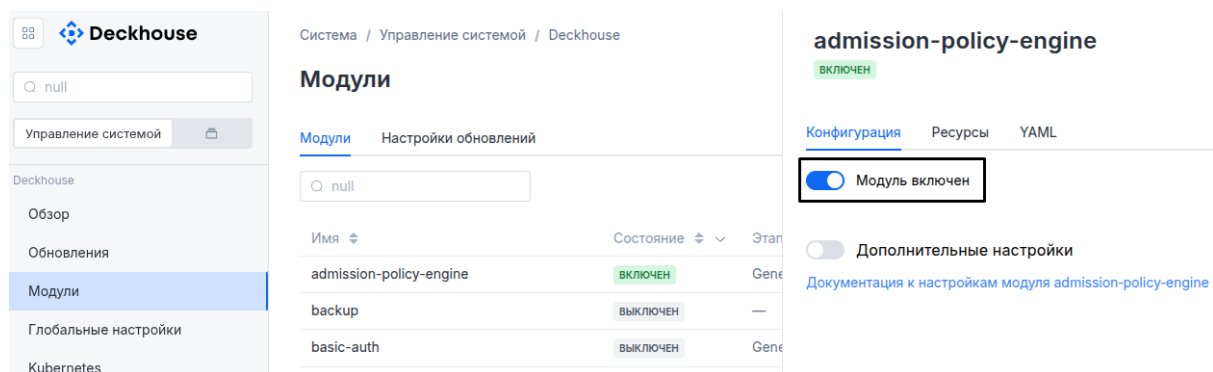
В модуле реализовано 11 алертов.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes-control-plane`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `monitoring-kubernetes-control-plane`:

```
d8 system module enable monitoring-kubernetes-control-plane
```

- **С помощью ModuleConfig monitoring-kubernetes-control-plane.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `monitoring-kubernetes-control-plane` (создайте его, при необходимости).

Пример манифеста для включения модуля `monitoring-kubernetes-control-plane`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes-control-plane
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes-control-plane`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `monitoring-kubernetes-control-plane` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-kubernetes-control-plane`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `monitoring-kubernetes-control-plane` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc monitoring-kubernetes-control-plane
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `monitoring-kubernetes-control-plane`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `monitoring-kubernetes-control-plane`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes-control-plane
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версии Kubernetes: 1.22.0 и выше.

К версии Deckhouse: 1.22.0 и выше.

К версиям других модулей:

- `prometheus`: любая версия.
- `control-plane-manager`: любая версия.

Мониторинг сетевого взаимодействия между всеми узлами кластера, а также (опционально) до дополнительных внешних узлов

Описание

Модуль мониторинга сетевого взаимодействия обеспечивает непрерывную проверку связности между всеми основными и, при необходимости, внешними узлами кластера.

Возможности и особенности модуля:

- автоматически проверяет доступность всех узлов кластера (и, опционально, внешних систем) с помощью ICMP (ping) — тестирование запускается каждые две секунды;
- все результаты экспортируются в формате метрик в систему мониторинга Prometheus;
- в комплекте — готовый дашборд для Grafana, где в реальном времени визуализируются текущая доступность, графики задержек и потенциальные проблемы с сетевой связностью;
- позволяет быстро выявлять узлы с деградировавшей связностью и ускоряет реакцию на инциденты.

Как работает

Модуль отслеживает любые изменения поля `.status.addresses` узла. Если они обнаружены, срабатывает хук, который собирает полный список имен узлов и их адресов и передает в `daemonSet`, который заново создает поды. Таким образом, `ping` проверяет всегда актуальный список узлов.

Модуль `monitoring-ping`: настройки

У модуля нет обязательных настроек.

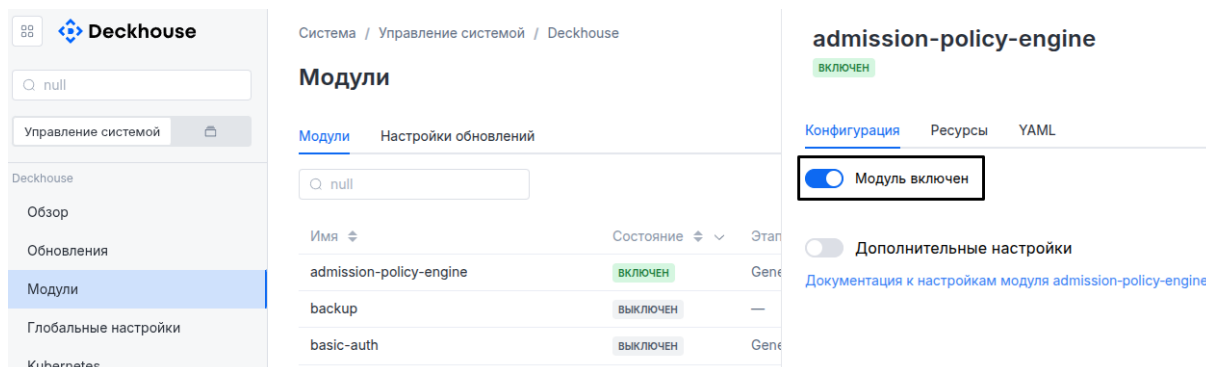
В модуле реализован 1 алерт.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-ping`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `monitoring-ping`:

```
d8 system module enable monitoring-ping
```

- **С помощью ModuleConfig `monitoring-ping`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `monitoring-ping` (создайте его, при необходимости).

Пример манифеста для включения модуля `monitoring-ping`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-ping`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `monitoring-ping` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `monitoring-ping`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `monitoring-ping` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc monitoring-ping
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `monitoring-ping`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `monitoring-ping`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `monitoring-kubernetes`: любая версия.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.externalTargets**
массив объектов

Список дополнительных хостов для мониторинга.

Пример:

```
- name: google-primary
  host: 8.8.8.8
- name: yaru
  host: ya.ru
- host: youtube.com
```

- **settings.externalTargets.host**
строка

Обязательный параметр

IP-адрес или host для мониторинга.

- **settings.externalTargets.name**
строка

Название записи мониторинга.

Модуль monitoring-ping: примеры конфигурации

Добавление дополнительных IP-адресов для мониторинга

Для добавления дополнительных IP-адресов мониторинга используйте параметр externalTargets модуля.

Пример конфигурации модуля:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings:
    externalTargets:
      - name: google-primary
        host: 8.8.8.8
      - name: yaru
        host: ya.ru
      - host: youtube.com
```

i Поле `name` используется в Grafana для отображения связанных данных. Если поле `name` не указано, используется обязательное поле `host`.

Модуль multitenancy-manager

 Стадия жизненного цикла модуля: Preview

Описание

Модуль позволяет создавать проекты в кластере Kubernetes. **Проект** — это изолированное окружение, в котором можно развернуть приложения.

Для чего это нужно?

Стандартный ресурс `Namespace`, который используется для логического разделения ресурсов в Kubernetes, не предоставляет необходимых функций, поэтому не является изолированным окружением:

- Потребление ресурсов подами по умолчанию не ограничено;
- Сетевое взаимодействие с другими подами по умолчанию работает из любой точки кластера;
- Неограниченный доступ к ресурсам узла: адресное пространство, сетевое пространство, смонтированные директории хоста.

Возможности настройки пространств имен `Namespace` не полностью соответствуют современным требованиям к разработке. По умолчанию для `Namespace` не включены следующие функции:

- Сборка логов;
- Аудит;
- Сканирование уязвимостей.

Функционал проектов позволяет решить эти проблемы.

 Модуль `secret-copier` не может использоваться совместно с модулем `multitenancy-manager`.

Преимущества модуля

Для администраторов платформы:

- **Единообразие:** Администраторы могут создавать проекты, используя один и тот же шаблон, что обеспечивает единообразие и упрощает управление.
- **Безопасность:** Проекты обеспечивают изоляцию ресурсов и политик доступа между различными проектами, что поддерживает безопасное многоадресное окружение.
- **Потребление ресурсов:** Администраторы могут легко устанавливать квоты на ресурсы и ограничения для каждого проекта, предотвращая избыточное использование ресурсов.

Для пользователей платформы:

- **Быстрый старт:** Разработчики могут запрашивать у администраторов проекты, созданные по готовым

шаблонам, что позволяет быстро начать разработку нового приложения.

- **Изоляция:** Каждый проект обеспечивает изолированное окружение, где разработчики могут развертывать и тестировать свои приложения без влияния на другие проекты.

Ограничения

Модуль работает только в рамках перечисленных ниже ограничений:

- Создание более одного пространства имён внутри проекта не предусматривается. Если требуется несколько пространств имён, создайте отдельный проект для каждого из них.
- Ресурсы шаблона применяются только к одному пространству имён, имя которого совпадает с именем проекта.

Внутренняя логика работы

Создание проекта

Для создания проекта используются ресурсы:

- ProjectTemplate — ресурс, который описывает шаблон проекта. При помощи него задается список ресурсов, которые будут созданы в проекте, а также схема параметров, которые можно передать при создании проекта;
- Project — ресурс, который описывает конкретный проект.

При создании ресурса Project из определенного ProjectTemplate происходит следующее:

1. Переданные параметры валидируются по OpenAPI-спецификации (параметр `parametersSchema.openAPIV3Schema` ресурса ProjectTemplate);
2. Выполняется рендеринг шаблона для ресурсов с помощью Helm. Значения для рендеринга берутся из параметра `parameters` ресурса Project;
3. Создается Namespace с именем, которое совпадает с именем Project;
4. По очереди создаются все ресурсы, описанные в шаблоне.



Внимание. При изменении шаблона проекта, все созданные проекты будут обновлены в соответствии с новым шаблоном.

Изоляция проекта

В основе проекта используется механизм изоляции ресурсов в рамках пространства имен (Namespace). Пространства имен позволяют группировать поды, сервисы, секреты и другие объекты, но не обеспечивают полноценной изоляции. Проект расширяет функциональность пространств имен, предлагая дополнительные инструменты для повышения уровня контроля и безопасности. Для управления уровнем изоляции проекта можно использовать возможности Kubernetes, например:

- Ресурсы контроля доступа (AuthorizationRule / RoleBinding) — позволяют управлять взаимодействием объектов внутри Namespace. Вы можете задавать правила и назначать роли, чтобы точно контролировать, кто и что может делать в вашем проекте.
- Ресурсы контроля использования нагрузки (ResourceQuota) — с их помощью можно задать лимиты на использование процессорного времени (CPU), оперативной памяти (RAM), а также количества объектов внутри Namespace. Это помогает избежать чрезмерной нагрузки и обеспечивает мониторинг за приложениями в рамках проекта.
- Ресурсы контроля сетевой связности (NetworkPolicy) — управляют входящим и исходящим сетевым трафиком в Namespace. Таким образом, можно настроить разрешенные подключения между подами, улучшить безопасность и управляемость сетевого взаимодействия в рамках проекта.

Эти инструменты можно комбинировать, чтобы настроить проект в соответствии с требованиями вашего приложения.

Настройки



Стадия жизненного цикла модуля: Preview

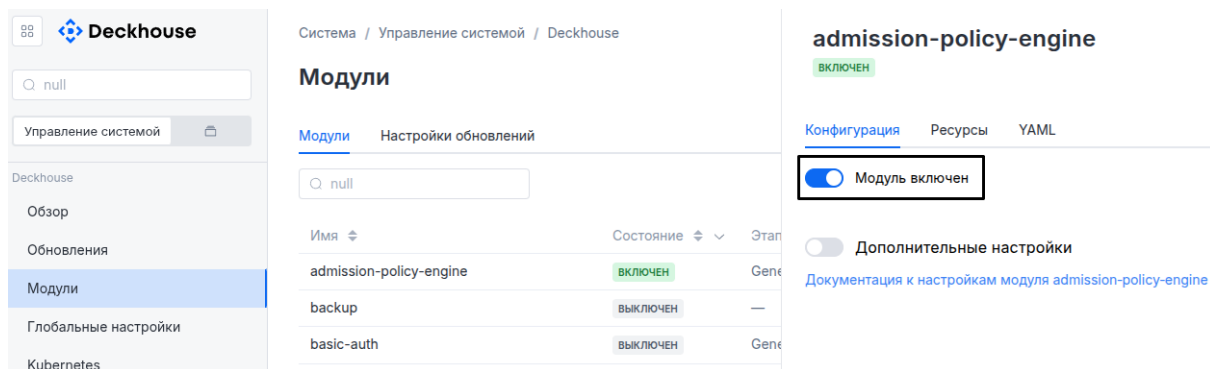
У модуля нет обязательных настроек.

Модуль **включен** по умолчанию в наборах модулей: Default, Managed. Модуль **выключен** по умолчанию в наборе модулей Minimal.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль multitenancy-manager, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `multitenancy-manager`:

```
d8 system module enable multitenancy-manager
```

- **С помощью ModuleConfig multitenancy-manager.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `multitenancy-manager` (создайте его, при необходимости).

Пример манифеста для включения модуля `multitenancy-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: multitenancy-manager
spec:
  enabled: true
```

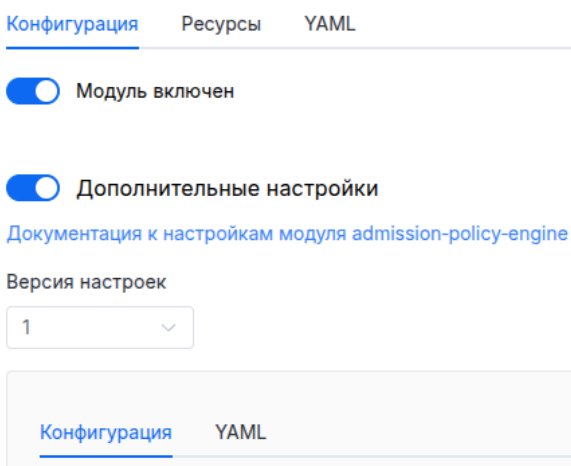
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `multitenancy-manager`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `multitenancy-manager` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `multitenancy-manager`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `multitenancy-manager` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc multitenancy-manager
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `multitenancy-manager`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `multitenancy-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: multitenancy-manager
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.allowNamespacesWithoutProjects**

булевый

Если эта опция выключена — запретите создавать пространства имен, которые не являются частью проекта.

По умолчанию: `true`

- **settings.highAvailability**

булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.



Стадия жизненного цикла модуля: Preview

Project

Scope: Cluster

- `v1alpha2`
- `v1alpha1`

- **spec**

объект

- **spec.description**

строка

Произвольное описание назначения проекта. Укажите пустую строку, если описание не требуется.

- **spec.parameters**

объект

Значения, передаваемые в шаблон ресурсов при создании проекта.

Перед созданием, значения согласовываются со схемой входных параметров шаблона ресурсов.

- **spec.projectTemplateName**

строка

Имя ресурса `ProjectTemplate`, который определяет, какие ресурсы будут созданы в проекте.

- **status**

объект

- **status.conditions**

массив объектов

- **status.conditions.lastProbeTime**

строка

- **status.conditions.lastTransitionTime**

строка

- **status.conditions.message**

строка

- **status.conditions.status**

строка

- **status.conditions.type**

строка

- **status.namespaces**

массив строк

Используемые пространства имен.

- **status.observedGeneration**

целочисленный

Последний generation проекта.

- **status.resources**

объект

Список срендеренных и пропущенных ресурсов.

- **status.state**

строка

Короткое описание состояния проекта, например, Error, Deployed, и т.д.

- **status.templateGeneration**

целочисленный

Последний generation шаблона.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

- **spec**

объект

- **spec.description**

строка

Произвольное описание назначения проекта. Укажите пустую строку, если описание не требуется.

- **spec.projectTypeName**

строка

Имя ресурса ProjectType, который определяет, какие ресурсы будут созданы в проекте.

- **spec.template**

объект

Значения, передаваемые в шаблон ресурсов при создании проекта. Перед созданием, значения согласовываются со схемой входных параметров шаблона ресурсов.

- **status**

объект

- **status.message**

строка

Сообщение, указывающее на причину появления текущего статуса.

- **status.state**

строка

Короткое описание состояния проекта, например, Ready, Error, Pending, и т.д.

- **status.sync**

булевый

Указывает на то, что параметры проекта были успешно применены после создания или изменения.

ProjectTemplate

Scope: Cluster

Version: v1alpha1

Описывает шаблон проекта.

На этот ресурс ссылается другой пользовательский ресурс — Project.

- **spec**

объект

- **spec.description**

строка

Произвольное описание назначения шаблона. Укажите пустую строку, если описание не требуется.

- **spec.parametersSchema**

объект

Обязательный параметр

Одна из схем параметров для валидации параметров проекта.

На данный момент поддерживается только схема OpenAPI v3.

- **spec.parametersSchema.openAPIV3Schema**

объект

Обязательный параметр

Спецификация OpenAPI для параметров шаблона.

Используется для проверки правильности параметров, передаваемых при создании нового проекта.

- **spec.resourcesTemplate**

строка

Обязательный параметр

Шаблон, определяющий ресурсы в формате `go template`, создаваемые при настройке нового проекта.

Шаблоны полностью совместимы со всеми функциями `helm`.

Узнайте больше о создании изолированной среды...

В шаблонах доступны следующие параметры:

- `{{ .projectName }}` — имя `Project`, для которого рендерится шаблон.
- `{{ .projectTemplateName }}` — имя `ProjectTemplate`.
- `{{ .parameters }}` — словарь пользовательских значений, описанных в параметре `.spec.parametersSchema` и определенных в параметре `.spec.parameters`.

i

Примечание! Указание полей `.metadata.namespace` для объектов является необязательным, так как в это поле автоматически устанавливается значение с именем созданного проекта.

- **status**

объект

- **status.message**

строка

Сообщение, указывающее на причину появления текущего статуса.

- **status.ready**

булевый

Готовность шаблона к использованию. Указывает на то, что шаблон успешно прошел проверку.

Примеры использования



Стадия жизненного цикла модуля: Preview

Шаблоны для проектов доступные по умолчанию

В Deckhouse Kubernetes Platform есть набор шаблонов для создания проектов:

- **default** — шаблон для базовых сценариев использования проектов:
 - ограничение ресурсов;
 - сетевая изоляция;
 - автоматические алерты и сбор логов;
 - выбор профиля безопасности;

Описание шаблона в GitHub.
- **secure** — включает все возможности шаблона **default**, а также дополнительные функции:
 - настройка допустимых для проекта UID/GID;
 - правила аудита обращения Linux-пользователей проекта к ядру;
 - сканирование запускаемых образов контейнеров на наличие известных уязвимостей (CVE).

Описание шаблона в GitHub.
- **secure-with-dedicated-nodes** — включает все возможности шаблона **secure**, а также дополнительные функции:
 - определение селектора узла для всех подов в проекте: если под создан, селектор узла пода будет автоматически **заменён** на селектор узла проекта;
 - определение стандартных tolerations для всех подов в проекте: если под создан, стандартные значения tolerations **добавляются** к нему автоматически.

Описание шаблона в GitHub.

Чтобы перечислить все доступные параметры для шаблона проекта, выполните команду:

```
d8 k get projecttemplates <ИМЯ_ШАБЛОНА_ПРОЕКТА> -o jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

Создание проекта

1. Для создания проекта создайте ресурс Project с указанием имени шаблона проекта в поле `.spec.projectTemplateName`.
2. В параметре `.spec.parameters` ресурса Project укажите значения параметров для секции `.spec.parametersSchema.openAPIV3Schema` ресурса ProjectTemplate.

Пример создания проекта с помощью ресурса Project из `default` ProjectTemplate представлен ниже:

```
apiVersion: deckhouse.io/v1alpha2
kind: Project
metadata:
  name: my-project
spec:
  description: This is an example from the Deckhouse documentation.
  projectTemplateName: default
  parameters:
    resourceQuota:
      requests:
        cpu: 5
        memory: 5Gi
        storage: 1Gi
      limits:
        cpu: 5
        memory: 5Gi
    networkPolicy: Isolated
    podSecurityProfile: Restricted
    extendedMonitoringEnabled: true
    administrators:
      - subject: Group
        name: k8s-admins
```

3. Для проверки статуса проекта выполните команду:

```
d8 k get projects my-project
```

Успешно созданный проект должен отображаться в статусе `Deployed` (синхронизирован). Если отображается статус `Error` (ошибка), добавьте аргумент `-o yaml` к команде (например, `d8 k get projects my-project -o yaml`) для получения более подробной информации о причине ошибки.

Автоматическое создание проекта для пространства имён

Для пространства имён возможно создать новый проект. Для этого пометьте пространство имён аннотацией `projects.deckhouse.io/adopt`. Например:

1. Создайте новое пространство имён:

```
d8 k create ns test
```

2. Пометьте его аннотацией:

```
d8 k annotate ns test projects.deckhouse.io/adopt=""
```

3. Убедитесь, что проект создался:

```
d8 k get projects
```

В списке проектов появится новый проект, соответствующий пространству имён:

NAME	STATE	PROJECT TEMPLATE	DESCRIPTION	AGE
deckhouse	Deployed	virtual	This is a virtual project	181d
default	Deployed	virtual	This is a virtual project	181d
test	Deployed	empty		1m

Шаблон созданного проекта можно изменить на существующий.



Обратите внимание, что при смене шаблона может возникнуть конфликт ресурсов: если в чарте шаблона прописаны ресурсы, которые уже присутствуют в пространстве имён, то применить шаблон не получится.

Создание собственного шаблона для проекта

Шаблоны проектов по умолчанию включают базовые сценарии использования и служат примером возможностей шаблонов.

Для создания своего шаблона:

1. Возьмите за основу один из шаблонов по умолчанию, например, `default`.
2. Скопируйте его в отдельный файл, например, `my-project-template.yaml` при помощи команды:

```
d8 k get projecttemplates default -o yaml > my-project-template.yaml
```

3. Отредактируйте файл `my-project-template.yaml`, внесите в него необходимые изменения.
`{% alert level="info" %}` Необходимо изменить не только шаблон, но и схему входных параметров под него.
 Шаблоны для проектов поддерживают все функции шаблонизации Helm. `{% endalert %}`
4. Измените имя шаблона в поле `.metadata.name`.
5. Примените полученный шаблон командой:

```
d8 k apply -f my-project-template.yaml
```

6. Проверьте доступность нового шаблона с помощью команды:

```
d8 k get projecttemplates <ИМЯ_НОВОГО_ШАБЛОНА>
```

Использование лейблов для управления ресурсами

При создании ресурсов в `ProjectTemplate` можно использовать специальные лейблы для управления поведением `multitenancy-manager` при обработке этих ресурсов:

Пропуск создания лейбла `heritage: multitenancy-manager`

По умолчанию все ресурсы, созданные из `ProjectTemplate`, получают лейбл `heritage: multitenancy-manager`. Он запрещает изменение ресурсов пользователями или любым контроллером, кроме `multitenancy-manager`. Если необходимо разрешить изменение ресурса (например, для совместимости с другими системами, или в случае реализации собственного контроля изменения создаваемых объектов), добавьте к ресурсу лейбл `projects.deckhouse.io/skip-heritage-label`.

Пример:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
labels:
  projects.deckhouse.io/skip-heritage-label: "true"
  app: my-app
data:
  key: value
```

В этом случае ресурс получит лейблы `projects.deckhouse.io/project` и `projects.deckhouse.io/project-template`, но не получит лейбл `heritage: multitenancy-manager`.

Исключение ресурсов из управления `multitenancy-manager`

Если необходимо исключить ресурс из управления `multitenancy-manager` (например, если он должен управляться вручную или другим контроллером), добавьте к ресурсу лейбл `projects.deckhouse.io/unmanaged`.

Пример:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
labels:
  projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Ресурсы с лейблом `projects.deckhouse.io/unmanaged`:

- Будут созданы **только один раз** при создании проекта;

- **Не будут обновляться** при последующих изменениях шаблона или обновлениях;
- Не будут отслеживаться в статусе проекта;
- Получат лейблы `projects.deckhouse.io/project` и `projects.deckhouse.io/project-template`, но **не получают** лейбл `heritage: multitenancy-manager`.



После того как ресурс помечен как `unmanaged`, он будет создан при первой установке, но не будет обновляться при изменении `ProjectTemplate`. После создания ресурс становится полностью независимым и должен управляться вручную.

Реализация валидации изменений объектов с помощью пользовательского лейбла

Модуль `multitenancy-manager` использует `ValidatingAdmissionPolicy` для защиты ресурсов с лейблом `heritage: multitenancy-manager` от ручных изменений.

Вы можете реализовать аналогичную валидацию для ресурсов с любым лейблом.

Как работает валидация в multitenancy-manager

Происходит валидация объектов с лейблом `heritage: multitenancy-manager`.

Для этого используются следующие ресурсы:

1. `ValidatingAdmissionPolicy` — определяет правила валидации:
 - Операции: `UPDATE` и `DELETE`;
 - Проверка: разрешены только операции от имени `service account` контроллера;
 - Применяется ко всем ресурсам и API группам.
2. `ValidatingAdmissionPolicyBinding` — определяет на какие объекты распространяется валидация:
 - Использует `namespaceSelector` и `objectSelector` для выбора ресурсов по лейблу `heritage: multitenancy-manager`.

Создание собственной валидации

Для реализации валидации для ресурсов с другим лейблом (например, `heritage: my-custom-label`):

1. Создайте файл с манифестами ресурсов `ValidatingAdmissionPolicy` и `ValidatingAdmissionPolicyBinding`:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:  ["*"]
        apiVersions: ["*"]
        operations: ["UPDATE", "DELETE"]
        resources:  ["*"]
        scope:      "*"
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-namespace:my-service-account"' # Замените на ваш service account.
      reason: Forbidden
      messageExpression: 'object.kind == "Namespace" ? "This resource is managed by "' +
        object.metadata.name + "' system. Manual modification is forbidden.'"
        : "This resource is managed by "' + object.metadata.namespace + "' system. Manual
        modification is forbidden.'"
    ---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    namespaceSelector:
      matchLabels:
        heritage: my-custom-label
    objectSelector:
      matchLabels:
        heritage: my-custom-label
```

2. Настройте параметры валидации:

- `policyName` — уникальное имя политики (должно совпадать с `Policy` и `Binding`);
- `request.userInfo.username` — имя `service account`, которому разрешено изменять ресурсы (замените на ваш `service account`);

- `heritage: my-custom-label` — значение лейбла `heritage` для ваших ресурсов (замените на ваше значение). Запрещено использование значение `multitenancy-manager`, `deckhouse`;
- `failurePolicy: Fail` — политика при ошибке валидации:
 - `Fail` — отклонять запрос при ошибке проверки,
 - `Ignore` — игнорировать ошибки валидации.
- `validationActions` — действия валидации:
 - `Deny` — отклонять неразрешенные операции,
 - `Audit` — записывать операции в аудит лог.

3. Примените политику:

```
d8 k apply -f my-validation-policy.yaml
```

4. Убедитесь, что ваши ресурсы имеют соответствующий лейбл `heritage`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label
```

Модуль namespace-configurator

Модуль `namespace-configurator` позволяет автоматически управлять аннотациями и лейблами на пространствах имён.

Чтобы автоматически включать новые пространства имён в мониторинг, добавьте лейбл `extended-monitoring.deckhouse.io/enabled=true`.

Как работает

Модуль следит за изменениями пространства имён и своей конфигурации:

- назначает лейблы и аннотации из конфигурации всем пространствам имён, подпадающим под шаблон `includeNames` и не подпадающим под шаблон `excludeNames`;
- игнорируются пространства имён с лейблами `heritage` с одним из значений `upmeter`, `deckhouse` или `multitenancy-manager`;
- при изменении конфигурации модуля лейблы и аннотации пространств имён будут переназначены согласно конфигурации.

Что нужно настроить?

Перечислите список желаемых лейблов и аннотаций, а также список шаблонов поиска пространств имён в конфигурации модуля.

Настройки

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `namespace-configurator`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `namespace-configurator`:

```
d8 system module enable namespace-configurator
```

- **С помощью ModuleConfig namespace-configurator.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `namespace-configurator` (создайте его, при необходимости).

Пример манифеста для включения модуля `namespace-configurator`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `namespace-configurator`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `namespace-configurator` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `namespace-configurator`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `namespace-configurator` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc namespace-configurator
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `namespace-configurator`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `namespace-configurator`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.configurations**

массив объектов

- **settings.configurations.annotations**

объект

Список аннотаций в формате `ключ: "значение"`.

Пример:

```
annotations:
  example: 'true'
```

- **settings.configurations.excludeNames**

массив строк

Список шаблонов для исключения пространств имен.

По умолчанию: `[]`

Пример:

```
excludeNames:
  - infra-test
```

- **settings.configurations.includeNames**

массив строк

Обязательный параметр

Список шаблонов для включения пространств имен.

По умолчанию: `[]`

Пример:

```
includeNames:
  - prod-.*
  - infra-.*
```

- **settings.configurations.labels**

объект

Список лейблов в формате `ключ: "значение"`.

Пример:

```
labels:
  istio-injection: 'true'
```

Примеры

Пример

Добавьте лейбл `extended-monitoring.deckhouse.io/enabled=true` и аннотацию `foo=bar` к каждому namespace, начинающемуся с `prod-` или `infra-`, за исключением `infra-test`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
      - annotations:
          foo: bar
        labels:
          extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
        - "^prod"
        - "^infra"
      excludeNames:
        - "infra-test"
```

Модуль network-gateway

Модуль для постройки сетевого шлюза из Kubernetes-узлов. Содержит в себе DHCP-сервер и настройщик SNAT.

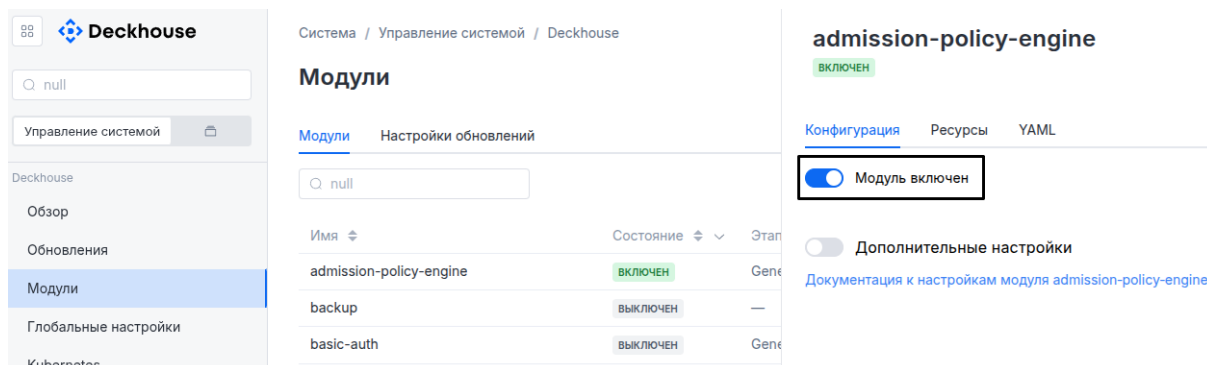
Настройки

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-gateway`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable network-gateway` для включения модуля, или `d8 system module disable network-gateway` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `network-gateway`:

```
d8 system module enable network-gateway
```

- **С помощью ModuleConfig network-gateway.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `network-gateway` (создайте его, при необходимости).

Пример манифеста для включения модуля `network-gateway`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-gateway
spec:
  enabled: true
```

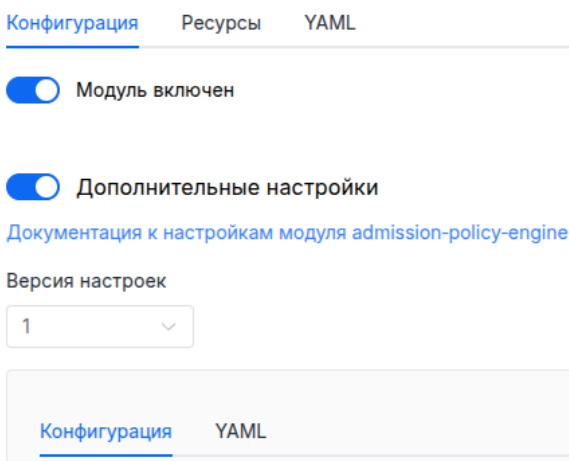
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-gateway`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `network-gateway` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-gateway`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `network-gateway` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc network-gateway
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `network-gateway`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `network-gateway`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-gateway
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.disableDHCP**
булевый

Отключает DHCP-сервер.

По умолчанию: `false`

- **settings.dns**
объект

Настройки, которые будут переданы клиентам через DHCP.

- **settings.dns.search**
массив строк

Список search-доменов.

Пример:

```
search:
- office.example.com
- srv.example.com
```

- **settings.dns.servers**
массив строк

Список DNS-серверов.

Пример:

```
servers:
- 4.2.2.2
- 8.8.8.8
```

- **settings.nodeSelector**
объект

Обязательный параметр

Селектор, который идентифицирует узлы, на которых будут настроены правила iptables и на которых будет работать DHCP-сервер.

Структура, аналогичная `spec.nodeSelector` пода Kubernetes. Поды инстанса унаследуют это поле как есть.

Пример:

```
nodeSelector:
  type: network-gateway
```

- **settings.publicAddress**

строка

Обязательный параметр

Адрес, на который будет заменен src-адрес исходящих из локальной сети пакетов.

Шаблон: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

Пример:

```
publicAddress: 10.220.203.240
```

- **settings.storageClass**

строка

Имя StorageClass, который будет использоваться для хранения DHCP lease.

Если значение не указано, используется StorageClass существующей PVC. Если PVC нет, будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск будет перезаказан, и все данные удалятся.

Если указать `false`, будет принудительно использоваться `emptyDir`.

Dnsmasq, используемый DHCP-сервером, включает механизмы защиты от дублирования IP-адресов при утере базы lease'ов, но лучше избегать ее потери.

- **settings.subnet**

строка

Обязательный параметр

Адрес локальной подсети, которую должен обслуживать шлюз.

На основе этого адреса будут сформированы DHCP-опции для выдачи клиентам:

- пул адресов — номера с 50 по последний;
- роутер — первый адрес в подсети.

Шаблон: `^([0-9]{1,3}\.){3}[0-9]{1,3}(\.([0-9]|[1-2][0-9]|3[0-2]))$`

Пример:

```
subnet: 192.168.42.0/24
```

- **settings.tolerations**

массив объектов

Toleration'ы для подов с DHCP и с настройщиками iptables.

Стандартный список toleration. Поды инстанса унаследуют это поле как есть.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Модуль network-policy-engine



Не используйте модуль, если включен модуль cilium, так как в нем уже есть функционал управления сетевыми политиками.

Модуль управления сетевыми политиками.

В Deckhouse выбран консервативный подход к организации сети, при котором используются простые сетевые бэкенды («чистый» CNI или flannel в режиме `host-gw`). Этот подход прост, надежен и показал себя с лучшей стороны.

Имплементация сетевых политик (`NetworkPolicy`) в Deckhouse также является простой и надежной системой, основанной на базе kube-router в режиме `Network Policy Controller` (`--run-firewall`). В этом случае kube-router транслирует сетевые политики `NetworkPolicy` в правила iptables, которые работают в любых инсталляциях вне

зависимости от облака или используемого CNI.

Модуль `network-policy-engine` разворачивает в namespace `d8-system` DaemonSet с kube-router в режиме поддержки network policies. В результате в Kubernetes-кластере включается полная поддержка Network Policies.

Поддерживаются следующие форматы описания политик:

- *networking.k8s.io/NetworkPolicy API*;
- *network policy V1/GA semantics*;
- *network policy beta semantics*.

Примеры использования.

Настройки

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-policy-engine`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left is a sidebar with navigation links: 'Обзор', 'Обновления', 'Модули' (selected), 'Глобальные настройки', and 'Kubernetes'. The main content area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. It contains a table of modules:

Имя	Состояние	Этап
admission-policy-engine	ВКЛЮЧЕН	Генерация
backup	ВЫКЛЮЧЕН	—
basic-auth	ВЫКЛЮЧЕН	Генерация

On the right, the configuration page for 'admission-policy-engine' is shown. It has tabs for 'Конфигурация', 'Ресурсы', and 'YAML'. Under 'Конфигурация', there is a toggle switch labeled 'Модуль включен' which is currently turned on. Below it is a toggle for 'Дополнительные настройки' which is turned off. A link to the documentation is also present.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `network-policy-engine`:

```
d8 system module enable network-policy-engine
```

- **С помощью ModuleConfig `network-policy-engine`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `network-policy-engine` (создайте его, при необходимости).

Пример манифеста для включения модуля `network-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-policy-engine
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-policy-engine`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

[Конфигурация](#) [Ресурсы](#) [YAML](#)

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

[Конфигурация](#) [YAML](#)

Вы также можете отредактировать объект ModuleConfig `network-policy-engine` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `network-policy-engine`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `network-policy-engine` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc network-policy-engine
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `network-policy-engine`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `network-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-policy-engine
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Примеры

Примеры network policies

Запретить обращаться снаружи к подам внутри namespace, но разрешить им обращаться внутри namespace и к внешним ресурсам

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: delete
spec:
  podSelector: {}
  egress:
  - {}
  ingress:
  - from:
    - podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Больше примеров можно посмотреть в этом репозитории.

Модуль node-local-dns

Описание модуля

Разворачивает кеширующий DNS-сервер на каждом узле кластера, экспортирует данные в `Prometheus` для удобного анализа работы DNS в кластере на доске Grafana.

Модуль состоит из оригинального `CoreDNS`, разворачиваемого в DaemonSet на все узлы, с добавлением алгоритма настройки сети и настройки правил iptables.

Назначение

Существует ряд проблем в стандартной работе DNS в Kubernetes, которые могут привести к неоправданному снижению ключевых показателей работы сервиса:

- Отсутствие кеширования запросов (в Linux из коробки нет кеша, соответственно, и в подах его тоже нет из коробки).
- Все DNS-запросы из контейнера влечут за собой сетевой запрос к кластерному DNS. Запросы к ресурсам в рамках одного узла все равно приводят к сетевым запросам.
- Запрос из пода сначала разрешается в кластерных DNS-зонах и только потом отправляется на внешние DNS-серверы. Например, запрос на `ya.ru` будет разрешаться сначала в кластерных зонах, таких как `cluster.local`, `svc.cluster.local`, `<namespace>.svc.cluster.local`, и только после получения отрицательных ответов, то есть по сути только с более чем второго раза, будет разрешаться правильно.

В случае появившихся небольших сетевых задержек качество сервиса может значительно деградировать из-за приведенных выше проблем.

Одно из решений — поставить DNS-сервер на каждый узел, для чего и предназначен настоящий модуль.

Внешние запросы также будут сначала пытаться разрешаться по цепочке внутренних зон, но только отсутствующие в кеше. При большой нагрузке (при большом количестве запросов на одни и те же записи в секунду, как это часто и бывает) такого поведения достаточно для значительного улучшения DNS-резолвинга.

Grafana dashboard

Дашборд `Kubernetes / DNS (node local)` отображает:

- общие графики (позволяют оценить в целом картину работы DNS);
- графики по узлам (позволяют глубже изучить найденную на общем графике проблему узла);
- графики по upstream (позволяют оценить работу кластерного DNS и серверов узлов, указанных в `/etc/resolv.conf`).

Как работает

На каждом узле модуль выполняет следующие настройки:

- Настраивает интерфейс с IP-адресом clusterIP сервиса `kube-dns`.
- Запускает кеширующий CoreDNS, который слушает на этом адресе.
- Добавляет правило в iptables, которое проверяет доступность локального сокета: если сокет открыт – трафик перенаправляется на локальный CoreDNS, если не открыт – обрабатывается стандартным механизмом Kubernetes для clusterIP.

Правило для обеспечения резервного пути обработки DNS (fallback):

```
-A PREROUTING -d <IP-адрес kube-dns> -m socket --nowildcard -j NOTRACK
```

Особенности конфигурации CoreDNS

Основные характеристики конфигурации CoreDNS:

- кеширование всех запросов;
- forward всех DNS-запросов в ClusterIP кластерного DNS.

Настройки



- Модуль работает со всеми CNI, но для корректной работы с `cni-cilium` необходимо выполнить ряд условий.
- По умолчанию, при использовании совместно с модулем `cni-simple-bridge` или `cni-flannel`, модуль `node-local-dns` **не работает** для запросов из `hostNetwork`. В этом случае все запросы уходят в модуль `kube-dns`. Вы можете указать адрес `169.254.20.10` в конфигурации пода, но тогда, в случае проблем в работе модуля `node-local-dns`, не будет работать *fallback* на модуль `kube-dns`.

В модуле реализовано 5 алертов.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-local-dns`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `node-local-dns`:

```
d8 system module enable node-local-dns
```

- **С помощью ModuleConfig `node-local-dns`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `node-local-dns` (создайте его, при необходимости).

Пример манифеста для включения модуля `node-local-dns`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-local-dns`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Вы также можете отредактировать объект ModuleConfig `node-local-dns` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-local-dns`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `node-local-dns` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc node-local-dns
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `node-local-dns`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `node-local-dns`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `kube-dns`: любая версия.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.disableIPv6**
булевый

Отключает преобразование DNS-адресов по протоколу IPv6, возвращая пустые ответы на запросы `AAAA` и `NXDOMAIN` на запросы обратного поиска `IPv6` в зоне `ip6.arpa`.

По умолчанию: `false`

- **settings.enableLogs**
булевый

Включает логирование запросов для экземпляров `node-local-dns`.

По умолчанию: `false`

- **settings.highAvailability**
булевый

Ручное управление режимом отказоустойчивости. В рамках данного модуля опция влияет на механизм кеширования DNS-записей на узлах:

- `true`: по истечении TTL у отдельной DNS-записи, `node-local-dns` продолжит возвращать “устаревшее” значение немедленно и параллельно с этим будет запрашивать свежее значение у вышестоящего `kube-dns`. Таким образом, при проблемах с узлами, на которых работает `kube-dns`, приложения продолжат получать ответы на DNS-запросы с устаревшим TTL, но без задержки.
- `false`: по истечении TTL у отдельной DNS-записи, `node-local-dns` продолжит возвращать “устаревшее” значение, но только после запроса к вышестоящему `kube-dns`. Таким образом, при проблемах с узлами, на которых работает `kube-dns`, приложения продолжат получать DNS-ответы с устаревшим TTL, но с возможной большой задержкой. Данный подход уместен в высокодинамичных окружениях, где часто пересоздаются `Service` и `Namespace`.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

Примеры

Пример настройки кастомного DNS в поде

```

apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx

```

Подробнее про настройку DNS.

Пример конфигурации

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  version: 1
  enabled: true
  settings:
    enableLogs: true

```

Управление узлами

Основные функции

Управление узлами осуществляется с помощью модуля `node-manager`, основные функции которого:

1. Управление несколькими узлами как связанной группой (**NodeGroup**):
 - Возможность определить метаданные, которые наследуются всеми узлами группы.
 - Мониторинг группы узлов как единой сущности (группировка узлов на графиках по группам, группировка алертов о недоступности узлов, алерты о недоступности N узлов или N% узлов группы).
2. Систематическое прерывание работы узлов — **Chaos Monkey**. Предназначено для верификации отказоустойчивости элементов кластера и запущенных приложений.
3. Задержка перезагрузки или выключения узла, пока на нём работают критичные поды.
4. Установка/обновление и настройка ПО узла (containerd, kubelet и др.), подключение узла в кластер:
 - Установка операционной системы (смотри список поддерживаемых ОС) вне зависимости от типа используемой инфраструктуры (в любом облаке или на любом железе).
 - Базовая настройка операционной системы (отключение автообновления, установка необходимых пакетов, настройка параметров журналирования и т. д.).
 - Настройка kubernetes-api-proxy (и системы автоматического обновления перечня upstream'ов) для балансировки запросов от узла (kubelet) по API-серверам.
 - Установка и настройка CRI containerd и Kubernetes, включение узла в кластер.
 - Управление обновлениями узлов и их простоем (disruptions):
 - Автоматическое определение допустимой минорной версии Kubernetes группы узлов на основании ее настроек (указанной для группы kubernetesVersion), версии по умолчанию для всего кластера и текущей действительной версии control plane (не допускается обновление узлов в опережение обновления control plane).
 - Из группы одновременно производится обновление только одного узла и только если все узлы группы доступны.
 - Два варианта обновлений узлов:
 - обычные — всегда происходят автоматически;
 - требующие disruption (например, обновление ядра, смена версии containerd, значительная смена версии kubelet и пр.) — можно выбрать ручной или автоматический режим. В случае, если разрешены автоматические disruptive-обновления, перед обновлением производится drain узла (можно отключить).
 - Мониторинг состояния и прогресса обновления.
5. Масштабирование кластера.
 - Автоматическое масштабирование.

Доступно при использовании поддерживаемых облачных провайдеров (подробнее) и недоступно для статических узлов. Облачный провайдер в автоматическом режиме может создавать или удалять виртуальные машины, подключать их к кластеру или отключать.
 - Поддержание желаемого количества узлов в группе.

Доступно как для облачных провайдеров, так и для статических узлов (при использовании Cluster API Provider Static).
6. Управление Linux-пользователями на узлах.
7. Поддержка GPU на узлах:
 - Автоматическое обнаружение и настройка NVIDIA GPU.
 - Различные режимы разделения ресурсов GPU на уровне NodeGroup: Exclusive, TimeSlicing, MIG.
 - Интеграция с мониторингом — в Grafana доступны готовые дашборды с ключевыми метриками GPU.

Управление узлами осуществляется через управление группой узлов (ресурс `NodeGroup`), где каждая такая группа выполняет определенные для нее задачи. Примеры групп узлов по выполняемым задачам:

- группы master-узлов;
- группа узлов маршрутизации HTTP(S)-трафика (front-узлы);
- группа узлов мониторинга;
- группа узлов приложений (worker-узлы) и т. п.

Узлы в группе имеют общие параметры и настраиваются автоматически в соответствии с параметрами группы. Deckhouse масштабирует группы, добавляя, исключая и обновляя ее узлы. Допускается иметь в одной группе как развернутые в облаке узлы типа `Static`, так и статические узлы (серверы `bare-metal`, виртуальные машины). Это позволяет получать узлы на физических серверах, которые могут масштабироваться за счет облачных узлов (гибридные кластеры).

Работа в облачной инфраструктуре осуществляется с помощью поддерживаемых облачных провайдеров. Если поддержки необходимой облачной платформы нет, возможно использование ее ресурсов в виде статических узлов.

Работа со статическими узлами (например, серверами `bare metal`) выполняется с помощью провайдера `CAPS` (`Cluster API Provider Static`).

Поддерживается работа со следующими сервисами `Managed Kubernetes` (может быть доступен не весь функционал сервиса):

- `Google Kubernetes Engine (GKE)`;
- `Elastic Kubernetes Service (EKS)`.

Типы узлов

Типы узлов, с которыми возможна работа в группах узлов (ресурс `NodeGroup`):

- `CloudEphemeral` — узлы автоматически заказываются, создаются и удаляются в настроенном облачном провайдере.
- `CloudPermanent` — узлы отличаются тем, что их конфигурация берется не из `custom resource nodeGroup`, а из специального ресурса `<PROVIDER>ClusterConfiguration` (например, `AWSClusterConfiguration` для AWS). Также важное отличие узлов в том, что для применения их конфигурации необходимо выполнить `dhctl converge` (запустить инсталлятор Deckhouse). Примером `CloudPermanent`-узла облачного кластера является master-узел кластера.
- `CloudStatic` — узел, созданный *вручную* (статический узел), размещенный в том же облаке, с которым настроена интеграция у одного из облачных провайдеров. На таком узле работает `CSI` и такой узел управляется `cloud-controller-manager`'ом. Объект `Node` кластера обогащается информацией о зоне и регионе, в котором работает узел. Также при удалении узла из облака соответствующий ему `Node`-объект будет удален в кластере.
- `Static` — статический узел, размещенный на сервере `bare metal` или виртуальной машине. В случае облака, такой узел не управляется `cloud-controller-manager`'ом, даже если включен один из облачных провайдеров. Подробнее про работу со статическими узлами...

Группировка узлов и управление группами

Группировка и управление узлами как связанной группой означает, что все узлы группы будут иметь одинаковые метаданные, взятые из кастомного ресурса `NodeGroup`.

Для групп узлов доступен мониторинг:

- с группировкой параметров узлов на графиках группы;
- с группировкой алертов о недоступности узлов;
- с алертами о недоступности `N` узлов или `N%` узлов группы и т. п.

Задержка перезагрузки или выключения узла, на котором работают критичные поды

С помощью сервиса `d8-shutdown-inhibitor` в `node-manager` реализован механизм задержки выключения или перезагрузки узла (далее просто «выключение узла»), если на нем запущены `stateful`-приложения или виртуальные машины (VM). Задержка срабатывает, если на узле запущены поды с лейблом `pod.deckhouse.io/inhibit-node-shutdown`. DKP откладывает выключение узла, чтобы пользователь смог перенести критичные поды на другой узел или завершить их работу штатным образом. Это полезно, например, для `stateful`-нагрузок и других случаев, когда необходимо избежать потери данных.

Принцип работы и особенности

Сервис `d8-shutdown-inhibitor` блокирует выключение узла до тех пор, пока на нём остаются поды в статусе `Running`, помеченные лейблом `pod.deckhouse.io/inhibit-node-shutdown`. Для блокировки используется состояние `GracefulShutdownPostpone`. Также на время блокировки выполняется `cordons` узла, чтобы `scheduler` не планировал на него новые поды. Kubelet отслеживает состояние `GracefulShutdownPostpone` узла и не начинает вытеснение помеченных лейблом подов, пока для этого состояния сохраняются `status: True` и `reason: PodsWithLabelAreRunningOnNode`. При наличии на узле подов с лейблом `pod.deckhouse.io/inhibit-node-shutdown` в консоль узла (wall) выводится соответствующее сообщение.

Особенности механизма в DKP

Задержка перезагрузки или выключения узла, на котором работают критичные поды, в DKP имеет следующие особенности:

- Лейбл необходимо явно задавать на каждый под, который должен блокировать выключение. Если поды создаются через `Deployment`, лейбл указывается в шаблоне пода.
- Механизм работает только с модифицированной версией kubelet, входящей в состав DKP. На стандартном Kubernetes аналогичное поведение отсутствует.

- Максимальное время блокировки (`InhibitDelayMaxSec`) составляет 3 суток. По истечении этого срока выключение может продолжиться независимо от наличия на узле подов с лейблом `pod.deckhouse.io/inhibit-node-shutdown`.
- Механизм не отменяет планирование disruptive-обновлений в `node-manager`. Он задерживает только фактическую перезагрузку узла в момент выключения.



Для принятия решения о блокировке выключения DKP дополнительно опрашивает NodeGroup. Если текущий узел принадлежит к группе `master` и является единственным `master`-узлом в кластере, блокировка выключения для него применена не будет.

Информация о включении механизма задержки представлена в разделе «Как отложить перезагрузку узла, пока на нём работают критичные поды».

Автоматическое развертывание, настройка и обновление узлов Kubernetes

Автоматическое развертывание (в *static/hybrid* — частично), настройка и дальнейшее обновление ПО работают на любых кластерах, независимо от его размещения в облаке или на bare metal.

Развертывание узлов Kubernetes

Deckhouse автоматически разворачивает узлы кластера, выполняя следующие **идемпотентные** операции:

- Настройку и оптимизацию операционной системы для работы с containerd и Kubernetes:
 - устанавливаются требуемые пакеты из репозитория дистрибутива;
 - настраиваются параметры работы ядра, параметры журналирования, ротация журналов и другие параметры системы.
- Установку требуемых версий containerd и kubelet, включение узла в кластер Kubernetes.
- Настройку `kubernetes-api-proxy` и обновление списка `upstream` для балансировки запросов от узла к Kubernetes API.

Поддержка актуального состояния узлов

Для поддержания узлов кластера в актуальном состоянии могут применяться два типа обновлений:

- **Обычные.** Такие обновления всегда применяются автоматически, и не приводят к остановке или перезагрузке узла.
- **Требующие прерывания** (disruption). Пример таких обновлений — обновление версии ядра или containerd, значительная смена версии kubelet и т. д. Для этого типа обновлений можно выбрать ручной или автоматический режим (секция параметров `disruptions`). В автоматическом режиме перед обновлением выполняется корректная приостановка работы узла (`drain`) и только после этого производится обновление.

В один момент времени производится обновление только одного узла из группы и только в том случае, когда все узлы группы доступны.

Модуль `node-manager` имеет набор встроенных метрик мониторинга, которые позволяют контролировать прогресс обновления, получать уведомления о возникающих во время обновления проблемах или о необходимости получения разрешения на обновление (ручное подтверждение обновления).

Работа с узлами в поддерживаемых облаках

У каждого поддерживаемого облачного провайдера существует возможность автоматического заказа узлов. Для этого необходимо указать требуемые параметры для каждого узла или группы узлов.

В зависимости от провайдера этими параметрами могут быть:

- тип узлов или количество ядер процессора и объем оперативной памяти;
- размер диска;
- настройки безопасности;
- подключаемые сети и др.

Создание, запуск и подключение виртуальных машин к кластеру выполняются автоматически.

Масштабирование узлов в облаке

Возможны два режима масштабирования узлов в группе:

- **Автоматическое масштабирование.**

При дефиците ресурсов, наличии подов в состоянии `Pending`, в группу будут добавлены узлы. При отсутствии нагрузки на один или несколько узлов, они будут удалены из кластера. При работе автомасштабирования учитывается приоритет группы (в первую очередь будет масштабироваться группа, у которой приоритет больше).

Чтобы включить автоматическое масштабирование узлов, необходимо указать разные ненулевые значения минимального и максимального количества узлов в группе.

- **Фиксированное количество узлов.**

В этом случае Deckhouse будет поддерживать указанное количество узлов (например, заказывая новые в случае выхода из строя старых узлов).

Чтобы указать фиксированное количество узлов в группе и отключить автоматическое масштабирование, необходимо указать одинаковые значения параметров `minPerZone` и `maxPerZone`.

Работа со статическими узлами

При работе со статическими узлами функции модуля `node-manager` выполняются со следующими ограничениями:

- **Отсутствует заказ узлов.** Непосредственное выделение ресурсов (серверов bare metal, виртуальных машин, связанных ресурсов) выполняется вручную. Дальнейшая настройка ресурсов (подключение узла к кластеру, настройка мониторинга и т.п.) выполняются полностью автоматически (аналогично узлам в облаке) или частично.
- **Отсутствует автоматическое масштабирование узлов.** Доступно поддержание в группе указанного количества узлов при использовании Cluster API Provider Static (параметр `staticInstances.count`). Т.е. Deckhouse будет пытаться поддерживать указанное количество узлов в группе, очищая лишние узлы и настраивая новые при необходимости (выбирая их из ресурсов `StaticInstance`, находящихся в состоянии `Pending`).

Настройка/очистка узла, его подключение к кластеру и отключение могут выполняться следующими способами:

- **Вручную**, с помощью подготовленных скриптов.

Для настройки сервера (BM) и ввода узла в кластер нужно загрузить и выполнить специальный bootstrap-скрипт. Такой скрипт генерируется для каждой группы статических узлов (каждого ресурса `NodeGroup`). Он находится в секрете `d8-cloud-instance-manager/manual-bootstrap-for-ИМЯ-NODEGROUP`. Пример добавления статического узла в кластер можно найти в FAQ.

Для отключения узла кластера и очистки сервера (виртуальной машины) нужно выполнить скрипт `/var/lib/bashible/cleanup_static_node.sh`, который уже находится на каждом статическом узле. Пример отключения узла кластера и очистки сервера можно найти в FAQ.

- **Автоматически**, с помощью Cluster API Provider Static.

Cluster API Provider Static (CAPS) подключается к серверу (BM) используя ресурсы `StaticInstance` и `SSHCredentials`, выполняет настройку, и вводит узел в кластер.

При необходимости (например, если удален соответствующий серверу ресурс `StaticInstance` или уменьшено количество узлов группы), Cluster API Provider Static подключается к узлу кластера, очищает его и отключает от кластера.

- **Вручную с последующей передачей узла под автоматическое управление** Cluster API Provider Static.

i Функциональность доступна начиная с версии Deckhouse 1.63.

Для передачи существующего узла кластера под управление CAPS необходимо подготовить для этого узла ресурсы `StaticInstance` и `SSHCredentials`, как при автоматическом управлении в пункте выше, однако ресурс `StaticInstance` должен дополнительно быть помечен аннотацией `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Cluster API Provider Static

Cluster API Provider Static (CAPS), это реализация провайдера декларативного управления статическими узлами (серверами bare metal или виртуальными машинами) для проекта Cluster API Kubernetes. По сути, CAPS это дополнительный слой абстракции к уже существующему функционалу Deckhouse по автоматической настройке и очистке статических узлов с помощью скриптов, генерируемых для каждой группы узлов (см. раздел Работа со статическими узлами).

CAPS выполняет следующие функции:

- настройка сервера bare metal (или виртуальной машины) для подключения к кластеру Kubernetes;
- подключение узла в кластер Kubernetes;
- отключение узла от кластера Kubernetes;
- очистка сервера bare metal (или виртуальной машины) после отключения узла из кластера Kubernetes.

CAPS использует следующие ресурсы (CustomResource) при работе:

- **StaticInstance.** Каждый ресурс `StaticInstance` описывает конкретный хост (сервер, BM), который управляется с помощью CAPS.
- **SSHCredentials.** Содержит данные SSH, необходимые для подключения к хосту (`SSHCredentials` указывается в параметре `credentialsRef` ресурса `StaticInstance`).
- **NodeGroup.** Секция параметров `staticInstances` определяет необходимое количество узлов в группе и фильтр множества ресурсов `StaticInstance` которые могут использоваться в группе.

CAPS включается автоматически, если в `NodeGroup` заполнена секция параметров `staticInstances`. Если в `NodeGroup` секция параметров `staticInstances` не заполнена, то настройка и очистка узлов для работы в этой группе выполняется *вручную* (см. примеры добавления статического узла в кластер и очистки узла), а не с помощью CAPS.

Схема работы со статическими узлами при использовании Cluster API Provider Static (CAPS) (практический пример добавления узла):

1. Подготовка ресурсов.

Перед тем, как отдать bare-metal-сервер или виртуальную машину под управление CAPS, может быть необходима предварительная подготовка, например:

- Подготовка системы хранения, добавление точек монтирования и т. п.;
- Установка специфических пакетов ОС;
- Настройка необходимой сетевой связности. Например, между сервером и узлами кластера;
- Настройка доступа по SSH на сервер, создание пользователя для управления с root-доступом через `sudo`. Хорошей практикой является создание отдельного пользователя и уникальных ключей для каждого сервера.

2. Создание ресурса SSHCredentials.

В ресурсе `SSHCredentials` указываются параметры, необходимые CAPS для подключения к серверу по SSH.

Один ресурс `SSHCredentials` может использоваться для подключения к нескольким серверам, но хорошей практикой является создание уникальных пользователей и ключей доступа для подключения к каждому серверу. В этом случае ресурс `SSHCredentials` также будет отдельный на каждый сервер.

3. Создание ресурса `StaticInstance`.

На каждый сервер (BM) в кластере создается отдельный ресурс `StaticInstance`. В нем указан IP-адрес для подключения и ссылка на ресурс `SSHCredentials`, данные которого нужно использовать при подключении.

Возможные состояния `StaticInstances` и связанных с ним серверов (BM) и узлов кластера:

- `Pending`. Сервер не настроен, и в кластере нет соответствующего узла.
- `Bootstrapping`. Выполняется процедура настройки сервера (BM) и подключения узла в кластер.
- `Running`. Сервер настроен, и в кластер добавлен соответствующий узел.
- `Cleaning`. Выполняется процедура очистки сервера и отключение узла из кластера.



Можно отдать существующий узел кластера, заранее введенный в кластер вручную, под управление CAPS, пометив его `StaticInstance` аннотацией `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

4. Создание ресурса `NodeGroup`.

В контексте CAPS в ресурсе `NodeGroup` нужно обратить внимание на параметр `nodeType` (должен быть `Static`) и секцию параметров `staticInstances`.

Секция параметров `staticInstances.labelSelector` определяет фильтр, по которому CAPS выбирает ресурсы `StaticInstance` для использования в группе. Фильтр позволяет использовать для разных групп узлов только определенные `StaticInstance`, а также позволяет использовать один `StaticInstance` в разных группах узлов. Фильтр можно не определять, чтобы использовать в группе узлов любой доступный `StaticInstance`.

Параметр `staticInstances.count` определяет желаемое количество узлов в группе. При изменении параметра, CAPS начинает добавлять или удалять необходимое количество узлов, запуская этот процесс параллельно.

В соответствии с данными секции параметров `staticInstances`, CAPS будет пытаться поддерживать указанное (параметр `count`) количество узлов в группе. При необходимости добавить узел в группу, CAPS выбирает соответствующий фильтру ресурс `StaticInstance` находящийся в статусе `Pending`, настраивает сервер (BM) и добавляет узел в кластер. При необходимости удалить узел из группы, CAPS выбирает `StaticInstance` находящийся в статусе `Running`, очищает сервер (BM) и удаляет узел из кластера (после чего, соответствующий `StaticInstance` переходит в состояние `Pending` и снова может быть использован).

Пользовательские настройки на узлах

Для автоматизации действий на узлах группы предусмотрен ресурс `NodeGroupConfiguration`. Ресурс позволяет выполнять на узлах bash-скрипты, в которых можно пользоваться набором команд `bashbooster`, а также позволяет использовать шаблонизатор `Go Template`. Это удобно для автоматизации таких операций, как:

- Установка и настройки дополнительных пакетов ОС.

Примеры:

- установка `kubectl`-плагина;
- настройка `containerd` с поддержкой `Nvidia GPU`.

- Обновление ядра ОС на конкретную версию.

Примеры:

- обновление ядра `Debian`;
- обновление ядра `CentOS`.

- Изменение параметров ОС.

Примеры:

- настройка параметра `sysctl`;
- добавление корневого сертификата.

- Сбор информации на узле и выполнение других подобных действий.

Ресурс `NodeGroupConfiguration` позволяет указывать приоритет выполняемым скриптам, ограничивать их выполнение определенными группами узлов и типами ОС.

Код скрипта указывается в параметре `content` ресурса. При создании скрипта на узле содержимое параметра `content` проходит через шаблонизатор `Go Template`, который позволяет встроить дополнительный уровень логики при генерации скрипта. При прохождении через шаблонизатор становится доступным контекст с набором динамических переменных.

Переменные, которые доступны для использования в шаблонизаторе:

- `.cloudProvider` (для групп узлов с `nodeType` `CloudEphemeral` или `CloudPermanent`) — массив данных облачного провайдера.

Пример данных...

```
cloudProvider:
  instanceClassKind: OpenStackInstanceClass
  machineClassKind: OpenStackMachineClass
  openstack:
    connection:
      authURL: https://cloud.provider.com/v3/
      domainName: Default
      password: p@ssw0rd
      region: region2
      tenantName: mytenantname
      username: mytenantusername
    externalNetworkNames:
      - public
    instances:
      imageName: ubuntu-24-04-cloud-amd64
      mainNetwork: kube
      securityGroups:
        - kube
      sshKeyPairName: kube
    internalNetworkNames:
      - kube
    podNetworkMode: DirectRoutingWithPortSecurityEnabled
  region: region2
  type: openstack
  zones:
    - nova
```

- `.cri` — используемый CRI (с версии Deckhouse 1.49 используется только `Containerd`).
- `.kubernetesVersion` — используемая версия Kubernetes.
- `.nodeUsers` — массив данных о пользователях узла, добавленных через ресурс `NodeUser`.

Пример данных...

```
nodeUsers:
- name: user1
  spec:
    isSudoer: true
    nodeGroups:
      - '*'
    passwordHash: PASSWORD_HASH
    sshPublicKey: SSH_PUBLIC_KEY
    uid: 1050
```

- `.nodeGroup` — массив данных группы узлов.

Пример данных...

```
nodeGroup:
  cri:
    type: Containerd
  disruptions:
    approvalMode: Automatic
  kubelet:
    containerLogMaxFiles: 4
    containerLogMaxSize: 50Mi
    resourceReservation:
      mode: "Off"
  kubernetesVersion: "1.31"
  manualRolloutID: ""
  name: master
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
    taints:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
  nodeType: CloudPermanent
  updateEpoch: "1699879470"
```

Пример использования переменных в шаблонизаторе:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

Пример использования команд bashbooster:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Ход выполнения скриптов можно увидеть на узле в журнале сервиса bashible с помощью команды:

```
journalctl -u bashible.service
```

Сами скрипты находятся на узле в директории `/var/lib/bashible/bundle_steps/`.

Сервис принимает решение о повторном запуске скриптов путем сравнения единой контрольной суммы всех файлов, расположенной по пути `/var/lib/bashible/configuration_checksum` с контрольной суммой размещенной в кластере kubernetes в секрете `configuration-checksums` namespace `d8-cloud-instance-manager`. Проверить контрольную сумму можно следующей командой:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -O yaml
```

Сравнение контрольных суммы сервис совершает каждую минуту.

Контрольная сумма в кластере изменяется раз в 4 часа, тем самым повторно запуская скрипты на всех узлах. Принудительный вызов исполнения bashible на узле можно произвести путем удаления файла с контрольной суммой скриптов с помощью следующей команды:

```
rm /var/lib/bashible/configuration_checksum
```

Особенности написания скриптов

При написании скриптов важно учитывать следующие особенности их использования в Deckhouse:

1. Скрипты в deckhouse выполняются раз в 4 часа или на основании внешних триггеров. Поэтому важно писать скрипты таким образом, чтобы они производили проверку необходимости своих изменений в системе перед выполнением действий, а не производили изменения каждый раз при запуске.
2. Существуют предзаготовленные скрипты которые производят различные действия в т.ч. установку и настройку сервисов. Важно учитывать это при выборе приоритета пользовательских скриптов. Например, если в скрипте планируется произвести перезапуск сервиса, то данный скрипт должен вызываться после скрипта установки сервиса. В противном случае он не сможет выполниться при развертывании нового узла.

Полезные особенности некоторых скриптов:

- `032_configure_containerd.sh` — объединяет конфигурационные файлы containerd и **перезапускает** сервис. Размещение конфигурационных файлов зависит от CRI на узлах кластера:
 - `/etc/containerd/conf.d/*.toml` — для containerd v1;
 - `/etc/containerd/conf2.d/*.toml` — для containerd v2.

Следует учитывать, что директории с конфигурационными файлами не создаются автоматически. Файлы в них нужно добавлять в скриптах с приоритетом менее 32.

Chaos Monkey

Инструмент (включается у каждой из `NodeGroup` отдельно), позволяющий систематически вызывать случайные прерывания работы узлов. Предназначен для проверки элементов кластера, приложений и инфраструктурных компонентов на реальную работу отказоустойчивости.

Мониторинг

Для групп узлов (ресурс `NodeGroup`) DKP экспортирует метрики доступности группы.

Какую информацию собирает Prometheus?

Все метрики групп узлов имеют префикс `d8_node_group` в названии, и лейбл с именем группы `node_group_name`.

Следующие метрики собираются для каждой группы узлов:

- `d8_node_group_ready` — количество узлов группы, находящихся в статусе `Ready`;
- `d8_node_group_nodes` — количество узлов в группе (в любом статусе);
- `d8_node_group_instances` — количество инстансов в группе (в любом статусе);
- `d8_node_group_desired` — желаемое (целевое) количество объектов `Machines` в группе;
- `d8_node_group_min` — минимальное количество инстансов в группе;
- `d8_node_group_max` — максимальное количество инстансов в группе;
- `d8_node_group_up_to_date` — количество узлов в группе в состоянии `up-to-date`;
- `d8_node_group_standby` — количество резервных узлов (см. параметр `standby`) в группе;

- `d8_node_group_has_errors` — единица, если в группе узлов есть какие-либо ошибки.

Настройки

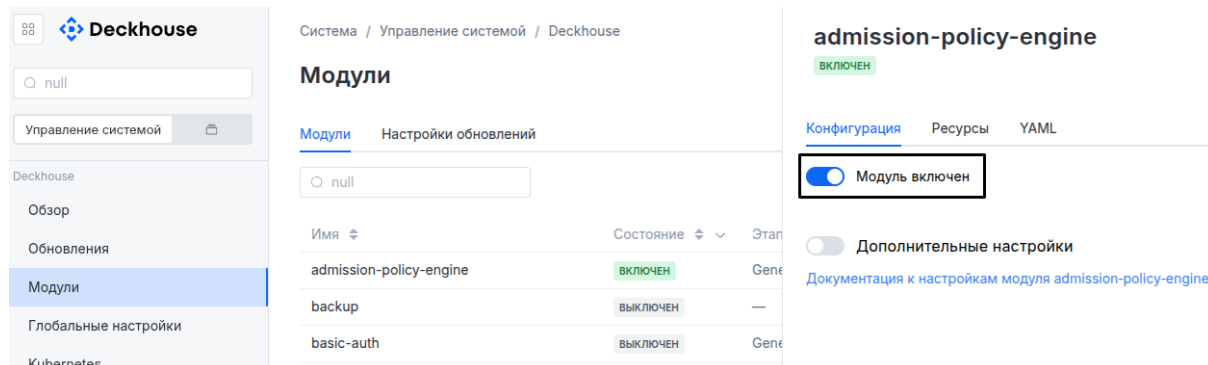
В модуле реализовано 37 алертов.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-manager`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `node-manager`:

```
d8 system module enable node-manager
```

- **С помощью ModuleConfig node-manager.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig node-manager` (создайте его, при необходимости).

Пример манифеста для включения модуля `node-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-manager`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1



Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `node-manager` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `node-manager`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `node-manager` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc node-manager
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `node-manager`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `node-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Удалите `allowedKubernetesVersions` и `allowedBundles`.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.capiEmergencyBrake**
булевый

Флаг отключения `CAPi` для случаев, когда вносятся потенциально опасные изменения.



Внимание. Используйте этот флаг только в том случае, если вы знаете, зачем это делаете, и четко понимаете последствия.

По умолчанию: `false`

Пример:

```
capiEmergencyBrake: false
```

- **settings.earlyOomEnabled**
булевый

Флаг отключения early OOM killer для случаев, когда его работа создаёт проблемы в нормальной работе узлов.

По умолчанию: `true`

Пример:

```
earlyOomEnabled: false
```

- **settings.instancePrefix**
строка

Префикс, который следует использовать при создании инстансов в cloud-провайдере.

Значение по умолчанию может вычисляться из данных структуры `ClusterConfiguration`, если кластер был установлен установщиком Deckhouse.

Пример:

```
instancePrefix: kube
```

- **settings.mcmEmergencyBrake**
булевый

Флаг отключения `machine-controller-manager` для случаев, когда вносятся потенциально опасные изменения.



Внимание. Используйте только в полном осознании зачем это делается.

По умолчанию: `false`

Пример:

```
mcmEmergencyBrake: false
```

Custom resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Instance

Scope: Cluster

Version: v1alpha1

Описывает независимый от реализации объект эфемерной машины.

- **status**
объект
 - **status.bootstrapStatus**
объект

Информация о процессе начальной настройки узла.
 - **status.bootstrapStatus.description**
строка

Описание процесса получения логов начальной настройки узла.
 - **status.bootstrapStatus.logsEndpoint**
строка

http эндпоинт для получения логов начальной настройки узла.
 - **status.classReference**
объект

Ссылка на объект `InstanceClass`. Уникален для каждого модуля `cloud-provider-*`.
 - **status.classReference.kind**
строка

Тип объекта (например, `OpenStackInstanceClass`). Тип объекта указан в документации соответствующего модуля облачного провайдера.

Допустимые значения: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`
 - **status.classReference.name**
строка

Имя нужного `InstanceClass`-объекта (например, `finland-medium`).
 - **status.currentStatus**
объект

Текущий статус объекта.
 - **status.currentStatus.lastUpdateTime**
строка

Время последнего обновления ствтуса.
 - **status.currentStatus.phase**
строка

Лейбл состояния машин в текущий момент времени.
 - **status.lastOperation**
объект

Последняя выполненная операция.
 - **status.lastOperation.description**
строка

Описание текущей операции.

- **status.lastOperation.lastUpdateTime**
строка
Время последней операции.
- **status.lastOperation.state**
строка
Состояние операции.
- **status.lastOperation.type**
строка
Тип операции.
- **status.machineRef**
объект
Ссылка на объект узла облачного провайдера.
 - **status.machineRef.apiVersion**
строка
Используемая версия API Kubernetes.
 - **status.machineRef.kind**
строка
Тип объекта.
 - **status.machineRef.name**
строка
Имя объекта.
 - **status.machineRef.namespace**
строка
Пространство имен, в котором находится объект.
- **status.nodeRef**
объект
Ссылка на объект узла kubernetes.
 - **status.nodeRef.name**
строка
Имя узла

InstanceTypesCatalog

Scope: Cluster

Version: v1alpha1

Содержит информацию о данных, обнаруженных в облаке.

- **instanceTypes**
массив объектов
Обязательный параметр
Список доступных типов виртуальных машин.
 - **instanceTypes.cpu**
строка или число
Обязательный параметр
Количество vCPU для данного типа виртуальной машины.
Шаблон: `^(\\+|-)?((([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))((( [KMGTPE]i) | [numKMGTPE] | ([eE] (\\+|-)?((([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+)))))?$`
 - **instanceTypes.memory**
строка или число
Обязательный параметр

Объем оперативной памяти для данного типа виртуальной машины.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))([ [KMGTPe]i) | [numkMGTPe] | ([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))))?$`

- **instanceTypes.name**

строка

Имя типа виртуальной машины.

- **instanceTypes.rootDisk**

строка или число

Размер корневого диска.

Шаблон: `^(\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))([ [KMGTPe]i) | [numkMGTPe] | ([eE](\\+|-)?([0-9]+(\\. [0-9]*)?)|(\\. [0-9]+))))?$`

NodeGroup

Short names: ng

Scope: Cluster

- v1
- v1alpha2
- v1alpha1

Описывает runtime-параметры группы узлов.

Пример:

```
# NodeGroup for cloud nodes in AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
  minPerZone: 1
  maxPerZone: 2
  classReference:
    kind: AWSInstanceClass
    name: test
  nodeTemplate:
    labels:
      tier: test
---
# NodeGroup for static nodes on bare metal servers (or VMs).
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

- **metadata**

объект

- **metadata.name**

строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Максимальная длина: 42

- **status**

- объект

- **status.conditionSummary**

- объект

- **status.conditionSummary.ready**

- строка

- Status of the condition summary.

- Допустимые значения: `True`, `False`

- **status.conditionSummary.statusMessage**

- строка

- Status message about group handling.

- **status.conditions**

- массив объектов

- **status.conditions.lastTransitionTime**

- строка

- Last time the condition transit from one status to another.

- **status.conditions.message**

- строка

- Human readable message indicating details about last transition.

- **status.conditions.status**

- строка

- Обязательный параметр

- Status of the condition, one of `True`, `False`.

- **status.conditions.type**

- строка

- Обязательный параметр

- Type of node group condition.

- **status.deckhouse**

- объект

- **status.deckhouse.observed**

- объект

- Contains last timestamp when the resource change was noted by the operator and its checksum.

- **status.deckhouse.observed.checkSum**

- строка

- The checksum of the last observed resource.

- **status.deckhouse.observed.lastTimestamp**

- строка

- Last timestamp when a change of the resource was noted by the operator.

- **status.deckhouse.processed**

- объект

- Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

- **status.deckhouse.processed.checkSum**

- строка

- The checksum of the last applied resource.

- **status.deckhouse.processed.lastTimestamp**

- строка

- Last timestamp when the resource was applied in the cluster.

- **status.deckhouse.synced**

- строка

True if last observed version of the resource was successfully applied in the cluster.

- **status.desired**
целочисленный

Number of desired machines in the group.

- **status.error**
строка

Error message about possible problems with the group handling.

- **status.instances**
целочисленный

Number of instances (in any state) in the group.

- **status.kubernetesVersion**
строка

The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

- **status.lastMachineFailures**
массив объектов

- **status.lastMachineFailures.lastOperation**
объект

- **status.lastMachineFailures.lastOperation.description**
строка

Last operation's description.

- **status.lastMachineFailures.lastOperation.lastUpdateTime**
строка

Timestamp of last status update for operation.

- **status.lastMachineFailures.lastOperation.state**
строка

Machine's operation state.

- **status.lastMachineFailures.lastOperation.type**
строка

Type of operation.

- **status.lastMachineFailures.name**
строка

Machine's name.

- **status.lastMachineFailures.ownerRef**
строка

Machine owner's name.

- **status.lastMachineFailures.providerID**
строка

Machine's ProviderID.

- **status.max**
целочисленный

Maximum amount of instances in the group.

- **status.min**
целочисленный

Minimal amount of instances in the group.

- **status.nodes**
целочисленный

Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
целочисленный

Number of ready Kubernetes nodes in the group.

- **status.standby**
целочисленный

Number of overprovisioned instances in the group.

- **status.upToDate**
целочисленный

Number of up-to-date nodes in the group.

- **spec**
объект

Обязательный параметр

- **spec.chaos**
объект

Настройки Chaos Monkey.

Пример:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

- **spec.chaos.mode**
строка

Режим работы Chaos Monkey:

- `DrainAndDelete` — при срабатывании делает узлу drain, затем удаляет его.
- `Disabled` — не трогает данную NodeGroup.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `DrainAndDelete`

- **spec.chaos.period**
строка

Интервал времени срабатывания Chaos Monkey.

Задается в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

По умолчанию: 6h

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.cloudInstances**
объект

Параметры заказа облачных виртуальных машин.

i

Внимание. Допустимо использовать только совместно с `nodeType: CloudEphemeral`.

- **spec.cloudInstances.classReference**
объект

Обязательный параметр

Ссылка на объект `InstanceClass`. Уникален для каждого модуля `cloud-provider-*`.

- **spec.cloudInstances.classReference.kind**
строка

Тип объекта (например, `OpenStackInstanceClass`). Тип объекта указан в документации соответствующего модуля облачного провайдера.

Допустимые значения: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`,

AzureInstanceClass, VCDInstanceClass, ZvirtInstanceClass,
DynamixInstanceClass, HuaweiCloudInstanceClass, DVPInstanceClass

■ **spec.cloudInstances.classReference.name**

строка

Имя нужного InstanceClass-объекта (например, finland-medium).

■ **spec.cloudInstances.maxPerZone**

целочисленный

Обязательный параметр

Максимальное количество инстансов в зоне. Проставляется как верхняя граница в cluster-autoscaler.

Допустимые значения: $0 \leq X$

■ **spec.cloudInstances.maxSurgePerZone**

целочисленный

Количество одновременно создаваемых инстансов при scale-up.

По умолчанию: 1

Допустимые значения: $0 \leq X$

■ **spec.cloudInstances.maxUnavailablePerZone**

целочисленный

Недоступное количество инстансов при RollingUpdate'e.

По умолчанию: 0

Допустимые значения: $0 \leq X$

■ **spec.cloudInstances.minPerZone**

целочисленный

Обязательный параметр

Минимальное количество инстансов в зоне.

Проставляется в объект MachineDeployment и в качестве нижней границы в cluster autoscaler.

Допустимые значения: $0 \leq X$

■ **spec.cloudInstances.priority**

целочисленный

Приоритет группы узлов.

При масштабировании кластера cluster autoscaler будет выбирать сначала группы узлов с установленным большим приоритетом. Если существует несколько групп узлов с одинаковым приоритетом, группа будет выбрана из них случайным образом. Скоростью переключения на заказ узлов из групп с меньшим приоритетом можно управлять (подробнее — в разделе «FAQ»).

Использование приоритетов может быть удобно, например, для предпочтения заказа более дешевых узлов (например, spot-инстансов) перед более дорогими.

■ **spec.cloudInstances.quickShutdown**

Параметр устарел

булевый

Снижает время drain'a CloudEphemeral-машин до 5 минут.

■ **spec.cloudInstances.standby**

строка или число

Количество резервных (*подогретых*) узлов в этой NodeGroup во всех зонах.

Резервный узел — это узел кластера, на котором резервируются ресурсы, доступные в любой момент для масштабирования. Наличие такого узла позволяет cluster autoscaler'у не ждать инициализации узла (которая может занимать несколько минут), а сразу размещать на нем нагрузку.

Значение может быть абсолютным (например, 2) или процентом желаемых узлов

(например, 10%). Если указан процент, абсолютное значение рассчитывается исходя из процента от максимального количества узлов (параметр `maxPerZone`), округленного в меньшую сторону, но не менее одного.

Шаблон: `^[0-9]+%?$`

■ **spec.cloudInstances.standbyHolder**

объект

Объем резервируемых ресурсов.

Используется для определения необходимости заказа резервных узлов.

■ **spec.cloudInstances.standbyHolder.notHeldResources**

Параметр устарел
объект

Устарело: параметр больше не используется. Используйте параметр `overprovisioningRate`.

Резервируемые ресурсы.

■ **spec.cloudInstances.standbyHolder.notHeldResources.cpu**

строка или число

Количество CPU.

Значение может быть абсолютным (например, 1) или в *millicore-формате* (например, 1500m).

Шаблон: `^[0-9]+m?$`

■ **spec.cloudInstances.standbyHolder.notHeldResources.memory**

строка или число

Количество памяти.

Значение может быть абсолютным в байтах (например, 128974848) или в Kubernetes-формате с суффиксами: G, Gi, M, Mi (например, 750Mi).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **spec.cloudInstances.standbyHolder.overprovisioningRate**

целочисленный

Процент резервируемых ресурсов, рассчитываемый от ресурсов узла в `NodeGroup`.

По умолчанию: 50

Допустимые значения: `1 <= X <= 80`

■ **spec.cloudInstances.zones**

массив строк

Переопределение перечня зон, в которых создаются инстансы.

Значение по умолчанию зависит от выбранного облачного провайдера и обычно соответствует всем зонам используемого региона.

Пример:

```
zones:
- Helsinki
- Espoo
- Tampere
```

○ **spec.cri**

объект

Параметры container runtime.

■ **spec.cri.containerd**

объект

Параметры работы containerd.

При настройке этих параметров `cri.type` должен быть `Containerd`.

- **spec.cri.containerd.maxConcurrentDownloads**

целочисленный

Максимальное количество параллельных потоков загрузки для каждой операции pull.

По умолчанию: 3

- **spec.cri.containerdV2**

объект

Параметры работы containerd V2.

При настройке этих параметров `cri.type` должен быть `ContainerdV2`.

При использовании `ContainerdV2` будет использоваться `CgroupsV2` (обеспечивает улучшенную безопасность и управление ресурсами).

Для использования `ContainerdV2` в качестве container runtime узлы в группе должны соответствовать следующим требованиям:

- поддержка `CgroupsV2`;
- ядро Linux версии 5.8 и новее;
- `systemd` версии 244 и новее;
- поддержка модуля ядра `erofs`.
- **spec.cri.containerdV2.maxConcurrentDownloads**

целочисленный

Максимальное количество параллельных потоков загрузки для каждой операции pull.

По умолчанию: 3

- **spec.cri.docker**

Параметр устарел
объект

Параметры настройки Docker.

- **spec.cri.docker.manage**

булевый

Автоматическое управление версией и параметрами Docker.

По умолчанию: `true`

- **spec.cri.docker.maxConcurrentDownloads**

целочисленный

Максимальное количество потоков одновременного скачивания Docker-образов.

По умолчанию: 3

- **spec.cri.notManaged**

объект

Настройки для CRI, установленных на узлах вручную.

- **spec.cri.notManaged.criSocketPath**

строка

Путь к сокету CRI.

- **spec.cri.type**

строка

Тип container runtime.

Если не указан, используется значение параметра `defaultCRI` ClusterConfiguration.



Внимание. Поддержка Docker считается устаревшей.

Допустимые значения: `Docker`, `Containerd`, `ContainerdV2`, `NotManaged`

◦ **spec.disruptions**

объект

Параметры обновлений, приводящих к возможному простою.

Пример:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
  windows:
    - from: '06:00'
      to: '08:00'
    days:
      - Tue
      - Sun
```

■ **spec.disruptions.approvalMode**

строка

Режим выдачи разрешения на disruptive-обновление (обновление, требующее прерывание работы узла):

- `Manual` — отключить автоматическую выдачу разрешений на disruptive-обновление. Если потребуется disruptive-обновление, появится специальный алерт. При выборе этого режима не происходит автоматический drain узла. Если требуется drain узла, сделайте это вручную перед установкой аннотации `update.node.deckhouse.io/disruption-approved=` на ресурс `node`. **Внимание!** Для группы узлов `master` режим выдачи разрешения всегда должен быть `Manual`, чтобы избежать проблем при drain'е узла.
- `Automatic` — автоматически выдавать разрешения на disruptive-обновление. При этом возможен автоматический drain узла, управляющийся настройкой `drainBeforeApproval`.
- `RollingUpdate` — в этом режиме будет создан **новый** узел с обновленными настройками, а старый узел будет удален. Разрешено только для облачных узлов.

Когда не используется режим `RollingUpdate`, при обновлении узел освобождается от нагрузки (drain), после чего обновляется (перезагружается) и вводится в работу. Обратите внимание, что в этом случае в кластере должно быть место для размещения нагрузки на время, пока обновляемый узел недоступен. В режиме `RollingUpdate` узел **заменяется** на обновленный, то есть на время обновления в кластере появляется дополнительный узел. В облачной инфраструктуре режим `RollingUpdate` удобен, например, если в кластере нет ресурсов для временного размещения нагрузки с обновляемого узла.

По умолчанию: `Automatic`

Допустимые значения: `Manual`, `Automatic`, `RollingUpdate`

■ **spec.disruptions.automatic**

объект

Дополнительные параметры для режима `Automatic`.

■ **spec.disruptions.automatic.drainBeforeApproval**

булевый

Выгон (draining) подов с узла перед выдачей разрешения на disruption.

Внимание! Данная настройка игнорируется (узлам будет выдано разрешение без предварительного выгона подов с узлов):

- для `nodeGroup master` с единственным узлом;
- для выделенного под запуск Deckhouse узла, если этот узел в группе узлов единственно рабочий (Ready).

По умолчанию: `true`

■ **spec.disruptions.automatic.windows**

массив объектов

Список окон disruption-обновлений узлов.

- **spec.disruptions.automatic.windows.days**

массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```

```
days: Wed
```

- **Элемент массива**

строка

День недели.

Допустимые значения: Mon, Tue, Wed, Thu, Fri, Sat, Sun

- **spec.disruptions.automatic.windows.from**

строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

- **spec.disruptions.automatic.windows.to**

строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

- **spec.disruptions.rollingUpdate**

объект

Дополнительные параметры для режима RollingUpdate.

- **spec.disruptions.rollingUpdate.windows**

массив объектов

Список окон disruption-обновлений узлов.

- **spec.disruptions.rollingUpdate.windows.days**

массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```

```
days: Wed
```

- **Элемент массива**

строка

День недели.

Допустимые значения: Mon, Tue, Wed, Thu, Fri, Sat, Sun

- **spec.disruptions.rollingUpdate.windows.from**
строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

- **spec.disruptions.rollingUpdate.windows.to**
строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

- **spec.fencing**
объект

Включает fencing-controller для группы узлов.

- **spec.fencing.mode**
строка

Обязательный параметр

Устанавливает режим работы fencing-controller.

Поддерживаемые режимы работы:

- **Watchdog** — использует модуль ядра watchdog для автоматической перезагрузки узлов, потерявших связь с кластером. Также в этом режиме fencing-agent отслеживает доступность узлов в кластере.

При включении `fencing` скрипт NodeGroupConfiguration загружает модуль ядра `softdog` с параметром `soft_margin`, равным значению `spec.fencing.watchdog.timeout` (по умолчанию 60 секунд), и параметром `soft_panic=1`.

Как работает fencing-agent:

- Агент открывает устройство `/dev/watchdog` и устанавливает на узле лейбл `node-manager.deckhouse.io/fencing-enabled`, а также лейбл `node-manager.deckhouse.io/fencing-mode` со значением `Watchdog`.
- Fencing-agent объединяется с агентами той же группы узлов в gossip-кластер (`memberlist`).
- Агент регулярно проверяет кворум (доступность большинства узлов группы):
 - если кворума нет (`quorum=false`), агент выполняет дополнительную проверку доступности Kubernetes API;
 - если кворум есть или Kubernetes API доступен, агент сбрасывает таймер `watchdog`;
 - если ни кворума, ни доступа к Kubernetes API нет, `watchdog` перестает сбрасываться, таймер истекает, что приводит к `kernel panic`.
- Агент учитывает аннотации обслуживания `node-manager.deckhouse.io/fencing-disable`, `update.node.deckhouse.io/approved` и `update.node.deckhouse.io/disruption-approved`, чтобы временно отключать `watchdog` на время плановых операций.

При включённом `fencing` автоматическая перезагрузка узла после `kernel panic` отключается на уровне ОС. Это исключает возврат узла с неопределённым состоянием до его корректной обработки (для статических узлов — вручную, для облачных узлов — через `cloud-provider-controller`).

Fencing-controller учитывает тип узла (`node.deckhouse.io/type`) при реакции на сбойный узел:

- для облачных узлов (`CloudEphemeral` , `CloudPermanent`) контроллер удаляет объект Node, после чего cloud-provider-controller пересоздаёт соответствующую виртуальную машину;
- для статических узлов (`Static` , `CloudStatic`) контроллер только вытесняет нагрузку, оставляя объект Node в кластере.

Подробнее о том, как механизм fencing обрабатывает разные типы узлов, можно почитать в разделе «FAQ».

Допустимые значения: `Watchdog`

■ **spec.fencing.watchdog**

объект

■ **spec.fencing.watchdog.timeout**

строка

Обязательный параметр

Длительность таймаута watchdog. Указывается в секундах, минутах или часах.

По умолчанию: `60s`

Шаблон: `^[0-9]+[smh]{1}$`

○ **spec.gpu**

объект

Доступно в редакциях: EE

Параметры GPU.

■ **spec.gpu.exclusive**

объект

Режим без разделения GPU.

■ **spec.gpu.mig**

объект

Параметры для стратегии `MIG`.

■ **spec.gpu.mig.customConfigs**

массив объектов

Кастомное разбиение MIG по индексам GPU.

Используется только при значении `partedConfig: custom`.

Представляет собой массив устройств (map по `index`), где каждый элемент указывает на GPU по его индексу и описывает создаваемые срезы.

Примеры:

```
customConfigs:
  profile: 3g.40gb
  count: 2
```

```
customConfigs:
  profile: 1g.10gb.me
  count: 1
```

■ **spec.gpu.mig.customConfigs.index**

целочисленный

Обязательный параметр

Индекс GPU на узле.

Допустимые значения: `0 <= X`

■ **spec.gpu.mig.customConfigs.slices**

массив объектов

Обязательный параметр

Список создаваемых MIG-срезов.

- **spec.gpu.mig.customConfigs.slices.count**
целочисленный

Обязательный параметр

Количество срезов выбранного профиля.

По умолчанию: 1

Допустимые значения: `1 <= X <= 7`

- **spec.gpu.mig.customConfigs.slices.profile**
строка

Обязательный параметр

Имя MIG-профиля.

Шаблон: `^[0-9]+g\[0-9\]+gb(\.me)?$`

Примеры:

```
profile: 3g.40gb
```

```
profile: 1g.10gb
```

```
profile: 1g.10gb.me
```

- **spec.gpu.mig.partedConfig**
строка

Имя используемой конфигурации MIG.

Поддерживаются стандартные профили вида `all-*` (одинаковое разбиение для всех GPU) или значение `custom`, позволяющее задать разбиение отдельно для каждой видеокарты.

По умолчанию: `all-1g.5gb`

- **spec.gpu.sharing**
строка

Стратегия разделения GPU:

- `Exclusive` — без разделения (каждый GPU полностью выделяется одному поду).
- `TimeSlicing` — разделение GPU по времени (несколько подов могут одновременно использовать один GPU, поочередно получая доступ к его вычислительным ресурсам).
- `MIG` — разделение одного GPU на несколько экземпляров (технология NVIDIA MIG, при наличии поддержки).

Допустимые значения: `TimeSlicing`, `MIG`, `Exclusive`

- **spec.gpu.timeSlicing**
объект

Параметры для стратегии `TimeSlicing`.

- **spec.gpu.timeSlicing.partitionCount**
число

Количество разделов (виртуальных GPU) на один физический GPU.

По умолчанию: 4

Допустимые значения: `1 <= X <= 100`

- **spec.kubelet**
объект

Параметры настройки kubelet.

- **spec.kubelet.containerLogMaxFiles**

целочисленный

Максимальное количество файлов журналов с учетом ротации.

По умолчанию: 4

Допустимые значения: $1 \leq X \leq 20$

- **spec.kubelet.containerLogMaxSize**

строка

Максимальный размер файла журнала до того, как он будет ротирован.

По умолчанию: 50Mi

Шаблон: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.maxPods**

целочисленный

Максимальное количество подов на узлах данной NodeGroup. Если явно не задан, по умолчанию задается в зависимости от значения параметра podSubnetNodeCIDRPrefix ClusterConfiguration:

- 120 подов для `podSubnetNodeCIDRPrefix ≥ 24`;
- 250 подов для `podSubnetNodeCIDRPrefix = 23`;
- 500 подов для `podSubnetNodeCIDRPrefix = 22`;
- 1000 подов для `podSubnetNodeCIDRPrefix ≤ 21`.

- **spec.kubelet.memorySwap**

объект

Конфигурация использования swap-памяти для kubelet.

- **spec.kubelet.memorySwap.limitedSwap**

объект

Настройки режима ограниченного использования swap. Требуются, если для swapBehavior указано значение LimitedSwap.

- **spec.kubelet.memorySwap.limitedSwap.size**

строка

Обязательный параметр

Размер создаваемого swap-файла в ГБ (например, "2G", "4G").

Шаблон: `^\d+G$`

Пример:

```
size: 1G
```

- **spec.kubelet.memorySwap.swapBehavior**

строка

Определяет, как используется swap-память на узле:

- NoSwap — swap отключен. Это традиционное поведение Kubernetes.
- LimitedSwap — kubelet может использовать swap-память в пределах, заданных в `limitedSwap.size`.

Подробнее об использовании swap-памяти на узлах можно прочитать в документации Kubernetes.

Допустимые значения: NoSwap, LimitedSwap

- **spec.kubelet.memorySwap.swappiness**

целочисленный

Определяет склонность ядра к использованию swap.

- 0 — минимальное использование (только для предотвращения OOM).
- 60 — сбалансированное поведение по умолчанию.
- 100 — агрессивное использование.

Значение устанавливается через параметр `sysctl vm.swappiness`.

По умолчанию: `60`

Допустимые значения: `0 <= X <= 100`

■ **spec.kubelet.resourceReservation**

объект

Управление резервированием ресурсов для системных служб на узле.

Больше информации в документации Kubernetes.

■ **spec.kubelet.resourceReservation.mode**

строка

Выбор режима резервирования системных ресурсов:

- `Off` — отключение резервирования системных ресурсов.
- `Auto` — автоматический подсчет резервирования через алгоритм, основанный на размере узла.
- `Static` — указание статических значений через параметр `static`.

Обратите внимание, что Deckhouse не использует выделенную `cgroup` для резервирования ресурсов (флаг `-system-reserved-cgroup` не используется).

По умолчанию: `Auto`

■ **spec.kubelet.resourceReservation.static**

объект

Параметры резервирования ресурсов в режиме `Static`.

■ **spec.kubelet.resourceReservation.static.cpu**

строка или число

Шаблон: `\d+[m]`

■ **spec.kubelet.resourceReservation.static.ephemeralStorage**

строка или число

Шаблон: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

■ **spec.kubelet.resourceReservation.static.memory**

строка или число

Шаблон: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

■ **spec.kubelet.rootDir**

строка

Путь к каталогу для файлов kubelet (volume mounts и т. д.).

По умолчанию: `/var/lib/kubelet`

■ **spec.kubelet.topologyManager**

объект

Управление менеджером топологии.

Менеджер топологии позволяет планировщику Kubernetes учитывать данные о топологии ресурсов при планировании подов.

В кластере с NUMA-узлами менеджер топологии может повысить производительность приложения, размещая контейнеры пода на том NUMA-узле, который эффективней работает с ресурсами.

Дополнительная информация в документации Kubernetes.

■ **spec.kubelet.topologyManager.enabled**

булевый

Включение использования менеджера топологий.

Если менеджер топологий включен, активируются режимы строгого выделения ресурсов для контейнеров подов класса `Guaranteed`. В настройках kubelet устанавливаются следующие значения параметров:

- `cpuManagerPolicy: static` — kubelet эксклюзивно закрепляет целые

- физические ядра за контейнером с целочисленным запросом CPU;
- `memoryManagerPolicy: Static` — kubelet закрепляет память контейнера за минимальным числом NUMA-узлов (предпочтительно одним).

По умолчанию: `false`

- **`spec.kubelet.topologyManager.policy`**
строка

Определение политики выравнивания ресурсов с учетом топологии узлов при планировании пода.

Поддерживаются четыре политики:

- `None` — выравнивание ресурсов не выполняется (используется по умолчанию).
- `BestEffort` — для каждого контейнера определяется предпочтительный NUMA-узел. Если предпочтительные узлы для планирования пода отсутствуют, то новый под будет запланирован на узел без учета данных менеджера топологии, а уже запущенный под продолжит работу.
- `Restricted` — для каждого контейнера пода определяется предпочтительный NUMA-узел. Если предпочтительные узлы для планирования пода отсутствуют, то новый под не будет запланирован на узел, а уже запущенный под будет остановлен.
- `SingleNumaNode` — для каждого контейнера пода определяется предпочтительный NUMA-узел. Если нет NUMA-узла, предпочтительного для **всех контейнеров пода**, то новый под не будет запланирован, а уже запущенный под будет остановлен.

Больше информации читайте в документации Kubernetes.

По умолчанию: `None`

Допустимые значения: `None`, `BestEffort`, `Restricted`, `SingleNumaNode`

- **`spec.kubelet.topologyManager.scope`**
строка

Определение гранулярности выравнивания ресурсов:

- `Container`. Вычисление предпочтительных ресурсов выполняется для контейнера. При вычислении не учитывается, что несколько контейнеров относятся к одному поду.
- `Pod`. Вычисление предпочтительных ресурсов выполняется для пода. Все контейнеры пода рассматриваются как единое целое при вычислении.

По умолчанию: `Container`

Допустимые значения: `Container`, `Pod`

- **`spec.nodeDrainTimeoutSecond`**
целочисленный

Максимальная продолжительность периода (в секундах), в течение которого контроллер будет пытаться переселить поды на новую машину в процессе выполнения `drain` на узле.

- **`spec.nodeTemplate`**
объект

Настройки `Node`-объектов в Kubernetes, которые будут добавлены после регистрации узла.

Пример:

```
nodeTemplate:
  labels:
    environment: production
    app: warp-drive-ai
  annotations:
    ai.fleet.com/discombobulate: 'true'
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
```

■ **spec.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

■ **spec.nodeTemplate.labels**

объект

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

■ **spec.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта Node.

i Доступны только поля `effect`, `key`, `value`.

Пример:

```
taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
```

■ **spec.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

■ **spec.nodeTemplate.taints.key**

строка

■ **spec.nodeTemplate.taints.value**

строка

○ **spec.nodeType**

строка

Обязательный параметр

Тип узлов, которые представляет эта группа:

- `CloudEphemeral` — узлы для этой группы будут автоматически создаваться (и удаляться) в настроенном облачном провайдере;
- `CloudPermanent` — узлы этой группы берутся из `ProviderClusterConfiguration` и создаются/удаляются через dhctl;
- `CloudStatic` — узел, созданный *вручную или любыми внешними инструментами*,

размещенный в том же облаке, с которым настроена интеграция у одного из облачных провайдеров. На таком узле работает CSI и такой узел управляется cloud-controller-manager'ом: объект Node автоматически обогащается информацией о зоне и регионе по данным, полученным от облака; при удалении узла из облака, соответствующий ему объект Node будет удален в Kubernetes;

- **Static** — статический узел, размещенный на физическом сервере или виртуальной машине. Узел не управляется cloud-controller-manager'ом, даже если включен один из облачных провайдеров.

Допустимые значения: `CloudEphemeral`, `CloudPermanent`, `CloudStatic`, `Static`

◦ **spec.operatingSystem** объект

Параметры операционной системы.

- **spec.operatingSystem.manageKernel**
Параметр устарел
булевый

Этот параметр не используется. Раньше он включал автоматическое управление ядром операционной системы.

По умолчанию: `true`

◦ **spec.staticInstances** объект

Параметры настройки машин статических узлов кластера.

- **spec.staticInstances.count**
целочисленный

Желаемое количество ресурсов StaticInstance (подпадающих под `labelSelector`), которые DKP автоматически настроит и добавит в кластер.

В NodeGroup со Static- и CloudStatic-узлами этот параметр задаёт ожидаемое количество узлов. DKP использует это значение для контроля состояния и автоматизации.

i **Внимание.** Этот параметр необходимо задавать явно. Значение по умолчанию — `0`, поэтому без изменения этого значения DKP не настроит ни один ресурс StaticInstance и не добавит в группу ни одного узла. Чтобы добавить статические узлы, укажите здесь нужное количество (больше `0`).

По умолчанию: `0`

Допустимые значения: `0 <= X`

- **spec.staticInstances.labelSelector**
объект

Настройка фильтра лейблов по ресурсам `staticInstances`.

Если указаны одновременно `matchExpressions` и `matchLabels`, учитываются оба фильтра (операция И).

Пустое значение `labelSelector` соответствует всем объектам. Нулевое — никаким.

i **Внимание.** Поле `labelSelector` можно добавить, но нельзя изменить после установки. Для изменения `labelSelector` создайте новую NodeGroup и переместите статические узлы в неё, изменив их лейблы.

- **spec.staticInstances.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений.

Итоговый результат — результат пересечения множеств, определяемых всеми фильтрами в списке (операция И).

Требования к селектору лейблов — он должен содержать значения, ключ и

оператор, связывающий ключ и значения.

- **spec.staticInstances.labelSelector.matchExpressions.key**
строка

Имя лейбла.
- **spec.staticInstances.labelSelector.matchExpressions.operator**
строка

Оператор сравнения.
- **spec.staticInstances.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.
 - **Элемент массива**
строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: `1..63`
- **spec.staticInstances.labelSelector.matchLabels**
объект

Фильтр на основе совпадения/несовпадения лейблов.

- **spec.update**

объект

- **spec.update.maxConcurrent**

строка или число

Максимальное количество одновременно обновляемых узлов.

Можно указать число узлов или процент от общего количества узлов в данной группе.

По умолчанию: `1`

Шаблон: `^[1-9][0-9]*%?$`

Описывает runtime-параметры группы узлов.

- **metadata**

объект

- **metadata.name**

строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Максимальная длина: `42`

- **status**

объект

- **status.conditionSummary**

объект

- **status.conditionSummary.ready**

строка

Status of the condition summary.

Допустимые значения: `True`, `False`

- **status.conditionSummary.statusMessage**

строка

Status message about group handling.

- **status.desired**

целочисленный

Number of desired machines in the group.

- **status.error**

строка

Error message about possible problems with the group handling.

- **status.instances**
целочисленный

Number of instances (in any state) in the group.
- **status.kubernetesVersion**
строка

The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.
- **status.lastMachineFailures**
массив объектов
 - **status.lastMachineFailures.lastOperation**
объект
 - **status.lastMachineFailures.lastOperation.description**
строка

Last operation's description.
 - **status.lastMachineFailures.lastOperation.lastUpdateTime**
строка

Timestamp of last status update for operation.
 - **status.lastMachineFailures.lastOperation.state**
строка

Machine's operation state.
 - **status.lastMachineFailures.lastOperation.type**
строка

Type of operation.
 - **status.lastMachineFailures.name**
строка

Machine's name.
 - **status.lastMachineFailures.ownerRef**
строка

Machine owner's name.
 - **status.lastMachineFailures.providerID**
строка

Machine's ProviderID.
- **status.max**
целочисленный

Maximum amount of instances in the group.
- **status.min**
целочисленный

Minimal amount of instances in the group.
- **status.nodes**
целочисленный

Number of Kubernetes nodes (in any state) in the group.
- **status.ready**
целочисленный

Number of ready Kubernetes nodes in the group.
- **status.standby**
целочисленный

Number of overprovisioned instances in the group.
- **status.upToDate**
целочисленный

Number of up-to-date nodes in the group.

- **spec**
объект

Обязательный параметр

- **spec.chaos**
объект

Настройки Chaos Monkey.

Пример:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

- **spec.chaos.mode**
строка

Режим работы Chaos Monkey:

- `DrainAndDelete` — при срабатывании делает узлу drain, затем удаляет его.
- `Disabled` — не трогает данную NodeGroup.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `DrainAndDelete`

- **spec.chaos.period**
строка

Интервал времени срабатывания Chaos Monkey.

Задается в виде строки с указанием часов и минут: `30m`, `1h`, `2h30m`, `24h`.

По умолчанию: `6h`

Шаблон: `^[0-9]+[mh]{1}$`

- **spec.cloudInstances**
объект

Параметры заказа облачных виртуальных машин.

i **Внимание.** Допустимо использовать только совместно с `nodeType: CloudEphemeral`.

- **spec.cloudInstances.classReference**
объект

Обязательный параметр

Ссылка на объект `InstanceClass`. Уникален для каждого модуля `cloud-provider-*`.

- **spec.cloudInstances.classReference.kind**
строка

Тип объекта (например, `OpenStackInstanceClass`). Тип объекта указан в документации соответствующего модуля облачного провайдера.

Допустимые значения: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`

- **spec.cloudInstances.classReference.name**
строка

Имя нужного `InstanceClass`-объекта (например, `finland-medium`).

- **spec.cloudInstances.maxPerZone**
целочисленный

Обязательный параметр

Максимальное количество инстансов в зоне.

Проставляется как верхняя граница в cluster-autoscaler.

Допустимые значения: `0 <= X`

■ **spec.cloudInstances.maxSurgePerZone**

целочисленный

Количество одновременно создаваемых инстансов при scale-up.

По умолчанию: `1`

Допустимые значения: `0 <= X`

■ **spec.cloudInstances.maxUnavailablePerZone**

целочисленный

Недоступное количество инстансов при RollingUpdate'e.

По умолчанию: `0`

Допустимые значения: `0 <= X`

■ **spec.cloudInstances.minPerZone**

целочисленный

Обязательный параметр

Минимальное количество инстансов в зоне.

Проставляется в объект `MachineDeployment` и в качестве нижней границы в cluster autoscaler.

При значении 0 для некоторых `InstanceClass` нужно задавать capacity. Более подробно можно узнать в описании нужных `InstanceClass`.

Допустимые значения: `0 <= X`

■ **spec.cloudInstances.standby**

строка или число

Количество резервных (*подогретых*) узлов в этой `NodeGroup` во всех зонах.

Резервный узел — это узел кластера, на котором резервируются ресурсы, доступные в любой момент для масштабирования. Наличие такого узла позволяет cluster autoscaler'у не ждать инициализации узла (которая может занимать несколько минут), а сразу размещать на нем нагрузку.

Значение может быть абсолютным (например, `2`) или процентом желаемых узлов (например, `10%`). Если указан процент, абсолютное значение рассчитывается исходя из процента от максимального количества узлов (параметр `maxPerZone`), округленного в меньшую сторону, но не менее одного.

Шаблон: `^[0-9]+%?$`

■ **spec.cloudInstances.standbyHolder**

объект

Объем резервируемых ресурсов.

Используется для определения необходимости заказа резервных узлов.

■ **spec.cloudInstances.standbyHolder.notHeldResources**

объект

Резервируемые ресурсы.

■ **spec.cloudInstances.standbyHolder.notHeldResources.cpu**

строка или число

Количество CPU.

Значение может быть абсолютным (например, `1`) или в *millicore-формате* (например, `1500m`).

Шаблон: `^[0-9]+m?$`

■ **spec.cloudInstances.standbyHolder.notHeldResources.memory**

строка или число

Количество памяти.

Значение может быть абсолютным в байтах (например, 128974848) или в Kubernetes-формате с суффиксами: G, Gi, M, Mi (например, 750Mi).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **spec.cloudInstances.zones**

массив строк

Переопределение перечня зон, в которых создаются инстансы.

Значение по умолчанию зависит от выбранного облачного провайдера и обычно соответствует всем зонам используемого региона.

Пример:

```
zones:
- Helsinki
- Espoo
- Tampere
```

○ **spec.cri**

объект

Параметры container runtime.

■ **spec.cri.containerd**

объект

Параметры работы containerd.

При настройке этих параметров `cri.type` должен быть `Containerd`.

■ **spec.cri.containerd.maxConcurrentDownloads**

целочисленный

Максимальное количество параллельных потоков загрузки для каждой операции pull.

По умолчанию: 3

■ **spec.cri.docker**

объект

Параметры настройки Docker.

i

Внимание. Docker считается **устаревшим**, не используйте его.

■ **spec.cri.docker.manage**

булевый

Автоматическое управление версией и параметрами Docker.

По умолчанию: `true`

■ **spec.cri.docker.maxConcurrentDownloads**

целочисленный

Максимальное количество потоков одновременного скачивания Docker-образов.

По умолчанию: 3

■ **spec.cri.notManaged**

объект

Настройки для CRI, установленных на узлах вручную.

■ **spec.cri.notManaged.criSocketPath**

строка

Путь к сокету CRI.

- **spec.cri.type**

строка

Тип container runtime.

Если не указан, используется значение параметра `defaultCRI` общих параметров кластера (ClusterConfiguration).



Внимание. Поддержка Docker считается устаревшей.

Допустимые значения: `Docker`, `Containerd`, `NotManaged`

- **spec.disruptions**

объект

Параметры обновлений, приводящих к возможному простое.

Пример:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
  windows:
    - from: '06:00'
      to: '08:00'
  days:
    - Tue
    - Sun
```

- **spec.disruptions.approvalMode**

строка

Режим выдачи разрешения на disruptive-обновление (обновление, требующее прерывание работы узла):

- `Manual` — отключить автоматическую выдачу разрешений на disruptive-обновление. Если потребуется disruptive-обновление, загорится специальный алерт. **Внимание!** Для группы узлов `master` режим выдачи разрешения всегда должен быть `Manual`, чтобы избежать проблем при drain'е узла.
- `Automatic` — автоматически выдавать разрешения на disruptive-обновление.
- `RollingUpdate` — в этом режиме будет создан **новый** узел с обновленными настройками, а старый узел будет удален. Разрешено только для облачных узлов.

Когда не используется режим `RollingUpdate`, при обновлении узел освобождается от нагрузки (drain), после чего обновляется (перезагружается) и вводится в работу. Обратите внимание, что в этом случае в кластере должно быть место для размещения нагрузки на время, пока обновляемый узел недоступен. В режиме `RollingUpdate` узел **заменяется** на обновленный, то есть на время обновления в кластере появляется дополнительный узел. В облачной инфраструктуре режим `RollingUpdate` удобен, например, если в кластере нет ресурсов для временного размещения нагрузки с обновляемого узла.

По умолчанию: `Automatic`

Допустимые значения: `Manual`, `Automatic`, `RollingUpdate`

- **spec.disruptions.automatic**

объект

Дополнительные параметры для режима `Automatic`.

- **spec.disruptions.automatic.drainBeforeApproval**

булевый

Выгон (draining) подов с узла перед выдачей разрешения на disruption.

Внимание! Данная настройка игнорируется (узлам будет выдано разрешение без предварительного выгона подов с узлов):

- для nodeGroup `master` с единственным узлом;

- для выделенного под запуск Deckhouse узла, если этот узел в группе узлов единственно рабочий (Ready).

По умолчанию: `true`

- **spec.disruptions.automatic.windows**
массив объектов

Список окон disruption-обновлений узлов.

- **spec.disruptions.automatic.windows.days**
массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```

```
days: Wed
```

- **Элемент массива**
строка

День недели.

Допустимые значения: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

- **spec.disruptions.automatic.windows.from**
строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

- **spec.disruptions.automatic.windows.to**
строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

- **spec.disruptions.rollingUpdate**
объект

Additional parameters for the `RollingUpdate` mode.

- **spec.disruptions.rollingUpdate.windows**
массив объектов

Time windows for node disruptive updates.

- **spec.disruptions.rollingUpdate.windows.days**
массив строк

Days of the week when node could be updated.

Примеры:

```
days: Mon
```



```
days: Wed
```

■ **Элемент массива**

строка

Day of the week.

Допустимые значения: Mon, Tue, Wed, Thu, Fri, Sat, Sun

■ **spec.disruptions.rollingUpdate.windows.from**

строка

Обязательный параметр

Start time of disruptive update window (UTC timezone).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

■ **spec.disruptions.rollingUpdate.windows.to**

строка

Обязательный параметр

End time of disruptive update window (UTC timezone).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

○ **spec.kubelet**

объект

Параметры настройки kubelet.

■ **spec.kubelet.containerLogMaxFiles**

целочисленный

Максимальное количество файлов журналов с учетом ротации.

i

Внимание. Параметр не влияет на работу, если тип CRI — Docker.

i

Внимание. Docker считается **устаревшим**, не используйте его.

По умолчанию: 4

Допустимые значения: `1 <= X <= 20`

■ **spec.kubelet.containerLogMaxSize**

строка

Максимальный размер файла журнала до того, как он будет ротирован.

Внимание! Параметр не влияет на работу, если тип CRI — Docker.

i

Внимание. Docker считается **устаревшим**, не используйте его.

По умолчанию: 50Mi

Шаблон: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

■ **spec.kubelet.maxPods**

целочисленный

..

максимальное количество подов на узлах данной `nodegroup`. Если явно не задан, по умолчанию задается в зависимости от значения параметра `podSubnetNodeCIDRPrefix` `ClusterConfiguration`:

- 120 подов для `podSubnetNodeCIDRPrefix ≥ 24`;
- 250 подов для `podSubnetNodeCIDRPrefix = 23`;
- 500 подов для `podSubnetNodeCIDRPrefix = 22`;
- 1000 подов для `podSubnetNodeCIDRPrefix ≤ 21`.

- **spec.kubelet.rootDir**

строка

Путь к каталогу для файлов kubelet (volume mounts и т. д.).

По умолчанию: `/var/lib/kubelet`

- **spec.nodeTemplate**

объект

Настройки `Node`-объектов в Kubernetes, которые будут добавлены после регистрации узла.

Пример:

```
nodeTemplate:
  labels:
    environment: production
    app: warp-drive-ai
  annotations:
    ai.fleet.com/discombobulate: 'true'
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
```

- **spec.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **spec.nodeTemplate.labels**

объект

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **spec.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта `Node`.

i Доступны только поля `effect`, `key`, `value`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **spec.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.nodeTemplate.taints.key**

строка

- **spec.nodeTemplate.taints.value**

строка

- **spec.nodeType**

строка

Обязательный параметр

Тип узлов, которые представляет эта группа:

- `Cloud` — узлы для этой группы будут автоматически создаваться (и удаляться) в настроенном облачном провайдере;
- `Static` — статический узел, размещенный на железном сервере или виртуальной машине. Узел не управляется `cloud-controller-manager`’ом, даже если включен один из облачных провайдеров;
- `Hybrid` — статический узел (созданный вручную или любыми внешними инструментами), размещенный в том же облаке, с которым настроена интеграция у одного из облачных провайдеров. На таком узле работает CSI и такой узел управляется `cloud-controller-manager`’ом: объект `Node` автоматически обогащается информацией о зоне и регионе по данным, полученным от облака; при удалении узла из облака соответствующий ему объект `Node` будет удален в Kubernetes.

Допустимые значения: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**

объект

Параметры операционной системы.

- **spec.operatingSystem.manageKernel**

булевый

Этот параметр не используется. Раньше он включал автоматическое управление ядром операционной системы.

По умолчанию: `true`

Описывает runtime-параметры группы узлов.

- **metadata**

объект

- **metadata.name**

строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Максимальная длина: `42`

- **status**

объект

- **status.conditionSummary**

объект

- **status.conditionSummary.ready**

строка

Status of the condition summary.

Допустимые значения: `True`, `False`

- **status.conditionSummary.statusMessage**

строка

Status message about group handling.

- **status.desired**
целочисленный

Number of desired machines in the group.
- **status.error**
строка

Error message about possible problems with the group handling.
- **status.instances**
целочисленный

Number of instances (in any state) in the group.
- **status.kubernetesVersion**
строка

The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.
- **status.lastMachineFailures**
массив объектов
 - **status.lastMachineFailures.lastOperation**
объект
 - **status.lastMachineFailures.lastOperation.description**
строка

Last operation's description.
 - **status.lastMachineFailures.lastOperation.lastUpdateTime**
строка

Timestamp of last status update for operation.
 - **status.lastMachineFailures.lastOperation.state**
строка

Machine's operation state.
 - **status.lastMachineFailures.lastOperation.type**
строка

Type of operation.
 - **status.lastMachineFailures.name**
строка

Machine's name.
 - **status.lastMachineFailures.ownerRef**
строка

Machine owner's name.
 - **status.lastMachineFailures.providerID**
строка

Machine's ProviderID.
- **status.max**
целочисленный

Maximum amount of instances in the group.
- **status.min**
целочисленный

Minimal amount of instances in the group.
- **status.nodes**
целочисленный

Number of Kubernetes nodes (in any state) in the group.
- **status.ready**
целочисленный

Number of ready Kubernetes nodes in the group.

- **status.standby**
целочисленный

Number of overprovisioned instances in the group.

- **status.upToDate**
целочисленный

Number of up-to-date nodes in the group.

- **spec**
объект

Обязательный параметр

- **spec.chaos**
объект

Настройки Chaos Monkey.

Пример:

```
chaos:
  mode: DrainAndDelete
  period: 24h
```

- **spec.chaos.mode**
строка

Режим работы Chaos Monkey:

- **DrainAndDelete** — при срабатывании делает узлу drain, затем удаляет его.
- **Disabled** — не трогает данную NodeGroup.

По умолчанию: **Disabled**

Допустимые значения: **Disabled**, **DrainAndDelete**

- **spec.chaos.period**
строка

Интервал времени срабатывания Chaos Monkey.

Задается в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

По умолчанию: **6h**

Шаблон: `^[0-9]+[mh]{1}$`

- **spec.cloudInstances**
объект

Параметры заказа облачных виртуальных машин.

i

Внимание. Допустимо использовать только совместно с **nodeType**: **CloudEphemeral**.

- **spec.cloudInstances.classReference**
объект

Обязательный параметр

Ссылка на объект **InstanceClass**. Уникален для каждого модуля **cloud-provider-***.

- **spec.cloudInstances.classReference.kind**
строка

Тип объекта (например, **OpenStackInstanceClass**). Тип объекта указан в документации соответствующего модуля облачного провайдера.

Допустимые значения: **OpenStackInstanceClass**, **GCPInstanceClass**, **VsphereInstanceClass**, **AWSInstanceClass**, **YandexInstanceClass**, **AzureInstanceClass**, **VCDInstanceClass**, **ZvirtInstanceClass**, **DynamixInstanceClass**, **HuaweiCloudInstanceClass**, **DVPInstanceClass**

■ spec.cloudInstances.classReference.name

строка

Имя нужного InstanceClass-объекта (например, `finland-medium`).**■ spec.cloudInstances.maxPerZone**

целочисленный

Обязательный параметр

Максимальное количество инстансов в зоне.

Проставляется как верхняя граница в `cluster-autoscaler`.Допустимые значения: `0 <= X`**■ spec.cloudInstances.maxSurgePerZone**

целочисленный

Количество одновременно создаваемых инстансов при scale-up.

По умолчанию: `1`Допустимые значения: `0 <= X`**■ spec.cloudInstances.maxUnavailablePerZone**

целочисленный

Недоступное количество инстансов при RollingUpdate'e.

По умолчанию: `0`Допустимые значения: `0 <= X`**■ spec.cloudInstances.minPerZone**

целочисленный

Обязательный параметр

Минимальное количество инстансов в зоне.

Проставляется в объект `MachineDeployment` и в качестве нижней границы в `cluster autoscaler`.

При значении 0 для некоторых InstanceClass нужно задавать capacity. Более подробно можно узнать в описании нужных InstanceClass.

Допустимые значения: `0 <= X`**■ spec.cloudInstances.standby**

строка или число

Количество резервных (*подогретых*) узлов в этой NodeGroup во всех зонах.

Резервный узел — это узел кластера, на котором резервируются ресурсы, доступные в любой момент для масштабирования. Наличие такого узла позволяет cluster autoscaler'у не ждать инициализации узла (которая может занимать несколько минут), а сразу размещать на нем нагрузку.

Значение может быть абсолютным (например, `2`) или процентом желаемых узлов (например, `10%`). Если указан процент, абсолютное значение рассчитывается исходя из процента от максимального количества узлов (параметр `maxPerZone`), округленного в меньшую сторону, но не менее одного.

Шаблон: `^[0-9]+%?$`**■ spec.cloudInstances.standbyHolder**

объект

Объем резервируемых ресурсов.

Используется для определения необходимости заказа резервных узлов.

■ spec.cloudInstances.standbyHolder.notHeldResources

объект

Резервируемые ресурсы.

■ spec.cloudInstances.standbyHolder.notHeldResources.cpu

строка или число

Количество CPU.

Значение может быть абсолютным (например, `1`) или в *millicore-формате* (например, `1500m`).

Шаблон: `^[0-9]+m?$`

■ **spec.cloudInstances.standbyHolder.notHeldResources.memory**

строка или число

Количество памяти.

Значение может быть абсолютным в байтах (например, `128974848`) или в Kubernetes-формате с суффиксами: `G`, `Gi`, `M`, `Mi` (например, `750Mi`).

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

■ **spec.cloudInstances.zones**

массив строк

Переопределение перечня зон, в которых создаются инстансы.

Значение по умолчанию зависит от выбранного облачного провайдера и обычно соответствует всем зонам используемого региона.

Пример:

```
zones:
- Helsinki
- Espoo
- Tampere
```

○ **spec.cri**

объект

Параметры container runtime.

■ **spec.cri.containerd**

объект

Параметры работы containerd.

При настройке этих параметров `cri.type` должен быть `Containerd`.

■ **spec.cri.containerd.maxConcurrentDownloads**

целочисленный

Максимальное количество параллельных потоков загрузки для каждой операции pull.

По умолчанию: `3`

■ **spec.cri.type**

строка

Container runtime type.

If not specified, the `defaultCRI` parameter value of ClusterConfiguration is used.

i **Note.** Docker support is deprecated.

Допустимые значения: `Docker`, `Containerd`, `NotManaged`

○ **spec.disruptions**

объект

Параметры обновлений, приводящих к возможному простоя.

Пример:

```

disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
  windows:
    - from: '06:00'
      to: '08:00'
    days:
      - Tue
      - Sun

```

■ **spec.disruptions.approvalMode**
строка

Режим выдачи разрешения на disruptive-обновление (обновление, требующее прерывания работы узла):

- **Manual** — отключить автоматическую выдачу разрешений на disruptive-обновление. Если потребуется disruptive-обновление, загорится специальный алерт. **Внимание!** Для группы узлов `master` режим выдачи разрешения всегда должен быть `Manual`, чтобы избежать проблем при `drain`'е узла.
- **Automatic** — автоматически выдавать разрешения на disruptive-обновление.
- **RollingUpdate** — в этом режиме будет создан **новый** узел с обновленными настройками, а старый узел будет удален. Разрешено только для облачных узлов.

Когда не используется режим `RollingUpdate`, при обновлении узел освобождается от нагрузки (`drain`), после чего обновляется (перезагружается) и вводится в работу. Обратите внимание, что в этом случае в кластере должно быть место для размещения нагрузки на время, пока обновляемый узел недоступен. В режиме `RollingUpdate` узел **заменяется** на обновленный, то есть на время обновления в кластере появляется дополнительный узел. В облачной инфраструктуре режим `RollingUpdate` удобен, например, если в кластере нет ресурсов для временного размещения нагрузки с обновляемого узла.

По умолчанию: `Automatic`

Допустимые значения: `Manual`, `Automatic`, `RollingUpdate`

■ **spec.disruptions.automatic**
объект

Дополнительные параметры для режима `Automatic`.

■ **spec.disruptions.automatic.drainBeforeApproval**
булевый

Выгон (`draining`) подов с узла перед выдачей разрешения на `disruption`.

Внимание. Данная настройка игнорируется (узлам будет выдано разрешение без предварительного выгона подов с узлов):

- для `nodeGroup master` с единственным узлом;
- для выделенного под запуск Deckhouse узла, если этот узел в группе узлов единственно рабочий (`Ready`).

По умолчанию: `true`

■ **spec.disruptions.automatic.windows**
массив объектов

Список окон `disruption`-обновлений узлов.

■ **spec.disruptions.automatic.windows.days**
массив строк

Дни недели, в которые применяется окно обновлений.

Примеры:

```
days: Mon
```



```
days: Wed
```

- **Элемент массива**

строка

День недели.

Допустимые значения: Mon, Tue, Wed, Thu, Fri, Sat, Sun

- **spec.disruptions.automatic.windows.from**

строка

Обязательный параметр

Время начала окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
from: '13:00'
```

- **spec.disruptions.automatic.windows.to**

строка

Обязательный параметр

Время окончания окна обновления (в часовом поясе UTC).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

```
to: '18:30'
```

- **spec.disruptions.rollingUpdate**

объект

Additional parameters for the RollingUpdate mode.

- **spec.disruptions.rollingUpdate.windows**

массив объектов

Time windows for node disruptive updates.

- **spec.disruptions.rollingUpdate.windows.days**

массив строк

Days of the week when node could be updated.

Примеры:

```
days: Mon
```

```
days: Wed
```

- **Элемент массива**

строка

Day of the week.

Допустимые значения: Mon, Tue, Wed, Thu, Fri, Sat, Sun

- **spec.disruptions.rollingUpdate.windows.from**

строка

Обязательный параметр

Start time of disruptive update window (UTC timezone).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

from: '13:00'

- **spec.disruptions.rollingUpdate.windows.to**

строка

Обязательный параметр

End time of disruptive update window (UTC timezone).

Шаблон: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Пример:

to: '18:30'

- **spec.docker**

объект

Параметры настройки Docker.

При настройке этих параметров `cri.type` должен быть `Docker`.

i

Внимание. `Docker` считается **устаревшим**, не используйте его.

- **spec.docker.manage**

булевый

Автоматическое управление версией и параметрами Docker.

По умолчанию: `true`

- **spec.docker.maxConcurrentDownloads**

целочисленный

Максимальное количество потоков одновременного скачивания Docker-образов.

По умолчанию: `3`

- **spec.kubelet**

объект

Параметры настройки kubelet.

- **spec.kubelet.containerLogMaxFiles**

целочисленный

Максимальное количество файлов журналов с учетом ротации.

i

Внимание. Параметр не влияет на работу, если тип CRI — `Docker`.

i

Внимание. `Docker` считается **устаревшим**, не используйте его.

По умолчанию: `4`

Допустимые значения: `1 <= X <= 20`

- **spec.kubelet.containerLogMaxSize**

строка

Максимальный размер файла журнала до того, как он будет ротирован.

Внимание! Параметр не влияет на работу, если тип CRI — `Docker`.

i

Внимание. `Docker` считается **устаревшим**, не используйте его.

По умолчанию: `50Mi`

Шаблон: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

■ **spec.kubelet.maxPods**

целочисленный

Максимальное количество подов на узлах данной `NodeGroup`. Если явно не задан, по умолчанию задается в зависимости от значения параметра `podSubnetNodeCIDRPrefix` `ClusterConfiguration`:

- 120 подов для `podSubnetNodeCIDRPrefix ≥ 24`;
- 250 подов для `podSubnetNodeCIDRPrefix = 23`;
- 500 подов для `podSubnetNodeCIDRPrefix = 22`;
- 1000 подов для `podSubnetNodeCIDRPrefix ≤ 21`.

■ **spec.kubelet.rootDir**

строка

Путь к каталогу для файлов kubelet (volume mounts и т. д.).

По умолчанию: `/var/lib/kubelet`

○ **spec.kubernetesVersion**

строка

Желаемая минорная версия Kubernetes.

По умолчанию соответствует глобально выбранной для кластера версии (см. документацию по установке) или, если таковая не определена, текущей версии `control plane`.

Допустимые значения: `1.31`, `1.32`, `1.33`, `1.34`, `1.35`

Пример:

```
kubernetesVersion: '1.32'
```

○ **spec.nodeTemplate**

объект

Настройки `Node`-объектов в Kubernetes, которые будут добавлены после регистрации узла.

Пример:

```
nodeTemplate:
  labels:
    environment: production
    app: warp-drive-ai
  annotations:
    ai.fleet.com/discombobulate: 'true'
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
```

■ **spec.nodeTemplate.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

■ **spec.nodeTemplate.labels**

объект

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **spec.nodeTemplate.taints**

массив объектов

Аналогично полю `.spec.taints` из объекта `Node`.

i

Доступны только поля `effect`, `key`, `value`.

Пример:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **spec.nodeTemplate.taints.effect**

строка

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.nodeTemplate.taints.key**

строка

- **spec.nodeTemplate.taints.value**

строка

- **spec.nodeType**

строка

Обязательный параметр

Тип узлов, которые представляет эта группа:

- `Cloud` — узлы для этой группы будут автоматически создаваться (и удаляться) в настроенном облачном провайдере;
- `Static` — статический узел, размещенный на железном сервере или виртуальной машине. Узел не управляется `cloud-controller-manager`’ом, даже если включен один из облачных провайдеров;
- `Hybrid` — статический узел (созданный вручную или любыми внешними инструментами), размещенный в том же облаке, с которым настроена интеграция у одного из облачных провайдеров. На таком узле работает CSI и такой узел управляется `cloud-controller-manager`’ом: объект `Node` автоматически обогащается информацией о зоне и регионе по данным, полученным от облака; при удалении узла из облака соответствующий ему объект `Node` будет удален в Kubernetes.

Допустимые значения: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**

Параметр устарел
объект

Параметры операционной системы.

- **spec.operatingSystem.manageKernel**

Параметр устарел
булевый

Этот параметр не используется. Раньше он включал автоматическое управление ядром операционной системы.

По умолчанию: `true`

- **spec.static**

объект

Параметры статического узла.

- **spec.static.internalNetworkCIDRs**

массив строк

CIDR подсети.

NodeGroupConfiguration

Short names: `ngc`

Scope: Cluster

Version: v1alpha1

Ресурс для выполнения bash-скриптов на узлах.

Подробнее в документации модуля.

- **spec**
объект

Обязательный параметр

- **spec.bundles**
массив строк

Обязательный параметр

Список bundle'ов, для которых будет выполняться скрипт. Для выбора всех bundle'ов нужно указать `'*'`.

Примеры:

```
bundles:
- ubuntu-lts
- centos-7
```

```
bundles:
- ubuntu-lts
```

```
bundles:
- "*"
```

- **Элемент массива**
строка

Допустимые значения: `ubuntu-lts`, `centos`, `debian`, `redos`, `rosa`, `astra`, `altlinux`, `opensuse`, `*`

- **spec.content**
строка

Обязательный параметр

Содержимое шага конфигурации в виде bash-скрипта.

Для генерации скрипта возможно использовать Go Template.

Получить список параметров, доступных для использования в шаблонах, можно из секрета `bashible-apiserver-context`:

```
d8 k -n d8-cloud-instance-manager get secrets bashible-apiserver-context -O
jsonpath='{.data.input\.yaml}'| base64 -d
```

Пример:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

Также в скрипте допустимо использовать подготовленные команды `bashbooster`.
Например:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Пример использования параметров проху, указанных в ресурсе ClusterConfiguration:

```
bb-set-proxy
curl -I https://google.com
bb-unset-proxy
```

```
test-function-with-proxy() {
  bb-set-proxy
  trap bb-unset-proxy RETURN
  curl -I https://google.com
}

test-function-with-proxy
```

- **spec.nodeGroups**

массив строк

Обязательный параметр

Список NodeGroup, к которым нужно применять шаг конфигурации. Для выбора всех NodeGroups нужно указать '*'.
Примеры:

```
nodeGroups:
- master
- worker
```

```
nodeGroups:
- worker
```

```
nodeGroups:
- "*"
```

- **spec.weight**

целочисленный

Очередность выполнения шага конфигурации.

По умолчанию: 100

NodeUser

Scope: Cluster

- v1
- v1alpha1

Описывает Linux-пользователей, которые будут созданы на всех узлах.

Домашний каталог пользователя создается в директории `/home/deckhouse/`.

- **spec**
объект

Обязательный параметр

- **spec.extraGroups**

массив строк

Список дополнительных групп, в которые должен быть включен пользователь.

Примеры:

```
extraGroups:
- docker
```

```
extraGroups:
- docker
- ftp
```

- **spec.isSudoer**
булевый

Определяет, будет ли пользователю разрешено sudo.

По умолчанию: `false`

Пример:

```
isSudoer: true
```

- **spec.nodeGroups**
массив строк

Список NodeGroup, к которым будет применяться настройка пользователя.

По умолчанию: `["*"]`

Примеры:

```
nodeGroups:
- master
- worker
```

```
nodeGroups:
- worker
```

```
nodeGroups:
- "*"
```

- **spec.passwordHash**
строка

Хеш пароля пользователя.

Формат соответствует хешам паролей, содержащихся в `/etc/shadow`. Его можно получить с помощью команды `openssl passwd -6`.

Пример:

```
passwordHash:
"$2a$10$F9ey7zW.sVliT224RFxpWeMsgz0.D9YRG54a8T36/K2MCiT41nzmC"
```

- **spec.sshPublicKey**
Параметр устарел
строка

Публичный SSH-ключ пользователя.

Обязательно должен быть указан либо `sshPublicKey`, либо `sshPublicKeys`.

Пример:

```
sshPublicKey: ssh-rsa AAABBB
```

- **spec.sshPublicKeys**
массив строк

Массив публичных SSH-ключей пользователя.

Обязательно должен быть указан либо `sshPublicKey`, либо `sshPublicKeys`.

Пример:

```
sshPublicKeys:
- ssh-rsa AAABBB
- cert-authority,principals="name" ssh-rsa BBBCCC
```

- **spec.uid**

число

Обязательный параметр

User ID пользователя на узлах.

Рекомендуем использовать значения `>= 1100` во избежание возможных конфликтов с пользователями, созданными вручную.

Неизменяемый в течение жизни ресурса параметр.

Допустимые значения: `1001 <= X`

Пример:

```
uid: 1100
```

- **status**

объект

Информация о статусе создания пользователя

- **status.errors**

объект

Список ошибок создания пользователя на каждом узле

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описывает Linux-пользователей, которые будут созданы на всех узлах.

Домашний каталог пользователя создается в директории `/home/deckhouse/`.

- **spec**

объект

Обязательный параметр

- **spec.extraGroups**

массив строк

Список дополнительных групп, в которые должен быть включен пользователь.

Примеры:

```
extraGroups:
- docker
```

```
extraGroups:
- docker
- ftp
```

- **spec.isSudoer**

булевый

Определяет, будет ли пользователю разрешено sudo.

По умолчанию: `false`

Пример:

```
isSudoer: true
```

- **spec.nodeGroups**

массив строк

Список NodeGroup, к которому будет применяться настройка пользователя.

По умолчанию: `["*"]`

Примеры:

```
nodeGroups:
- master
- worker
```

```
nodeGroups:
- worker
```

```
nodeGroups:
- "*"
```

- **spec.passwordHash**

строка

Хеш пароля пользователя.

Формат соответствует хешам паролей, содержащихся в `/etc/shadow`. Его можно получить с помощью команды `openssl passwd -6`.

Пример:

```
passwordHash:
"$2a$10$F9ey7zW.sVliT224RFxpWeMsgz0.D9YRG54a8T36/K2MCiT41nzmC"
```

- **spec.sshPublicKey**

Параметр устарел
строка

Публичный SSH-ключ пользователя.

Обязательно должен быть указан либо `sshPublicKey`, либо `sshPublicKeys`.

Пример:

```
sshPublicKey: ssh-rsa AAABBB
```

- **spec.sshPublicKeys**

массив строк

Массив публичных SSH-ключей пользователя.

Обязательно должен быть указан либо `sshPublicKey`, либо `sshPublicKeys`.

Пример:

```
sshPublicKeys:
- ssh-rsa AAABBB
- cert-authority,principals="name" ssh-rsa BBBCCC
```

- **spec.uid**

число

Обязательный параметр

User ID пользователя на узлах.

Рекомендуем использовать значения `>= 1100` во избежание возможных конфликтов с пользователями, созданными вручную.

Неизменяемый в течение жизни ресурса параметр.

Допустимые значения: `1001 <= X`

Пример:

uid: 1100

SSHCredentials

Scope: Cluster

- v1alpha2
- v1alpha1

Содержит данные, необходимые Cluster API Provider Static (CAPS) для подключения по протоколу SSH. CAPS подключается к серверу (виртуальной машине), описанному в ресурсе StaticInstance для управления его состоянием.

Ссылка на этот ресурс указывается в параметре credentialsRef ресурса StaticInstance.

- **apiVersion**
строка

Версия API объекта.

- **kind**
строка

Тип объекта.

- **metadata**
объект

- **spec**
объект

Желаемое состояние объекта SSHCredentials.

- **spec.privateSSHKey**
строка

Закрытый ключ SSH в формате PEM, закодированный в Base64.

Если у пользователя нет прав на просмотр конфиденциальных данных SSHCredentials, API возвращает `<omitted>` вместо фактического значения.

- **spec.sshExtraArgs**
строка

Список дополнительных параметров для SSH-клиента (`openssh`).

Примеры:

```
sshExtraArgs: "-vvv"
```

```
sshExtraArgs: "-c chacha20-poly1305@openssh.com"
```

```
sshExtraArgs: "-c aes256-gcm@openssh.com"
```

```
sshExtraArgs: "-m umac-64-etm@openssh.com"
```

```
sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
```

- **spec.sshPort**
целочисленный

Порт для подключения по SSH.

По умолчанию: `22`

Допустимые значения: `1 <= X <= 65535`

- **spec.sudoPasswordEncoded**
строка

Пароль пользователя для использования `sudo`, закодированный в Base64.

Если у пользователя нет прав на просмотр конфиденциальных данных SSHCredentials, API возвращает `<omitted>` вместо фактического значения.

- **spec.user**
строка

Обязательный параметр

Имя пользователя для подключения по SSH.

Содержит данные, необходимые Cluster API Provider Static (CAPS) для подключения по протоколу SSH. CAPS подключается к серверу (виртуальной машине), описанному в ресурсе StaticInstance для управления его состоянием.

Ссылка на этот ресурс указывается в параметре credentialsRef ресурса StaticInstance.

- **apiVersion**
строка

Версия API объекта.

- **kind**
строка

Тип объекта.

- **metadata**
объект

- **spec**
объект

Желаемое состояние объекта SSHCredentials.

- **spec.privateSSHKey**
строка

Закрытый ключ SSH в формате PEM, закодированный в Base64.

Если у пользователя нет прав на просмотр конфиденциальных данных SSHCredentials, API возвращает `<omitted>` вместо фактического значения.

- **spec.sshExtraArgs**
строка

Список дополнительных параметров для SSH-клиента (`openssh`).

Примеры:

```
sshExtraArgs: "-vvv"
```

```
sshExtraArgs: "-c chacha20-poly1305@openssh.com"
```

```
sshExtraArgs: "-c aes256-gcm@openssh.com"
```

```
sshExtraArgs: "-m umac-64-etm@openssh.com"
```

```
sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
```

- **spec.sshPort**
целочисленный

Порт для подключения по SSH.

По умолчанию: `22`

Допустимые значения: `1 <= X <= 65535`

- **spec.sudoPassword**
строка

Пароль пользователя для использования `sudo`.

Если у пользователя нет прав на просмотр конфиденциальных данных SSHCredentials,

API возвращает `<omitted>` вместо фактического значения.

- **spec.user**
строка

Обязательный параметр

Имя пользователя для подключения по SSH.

StaticInstance

Scope: Cluster

- v1alpha2
- v1alpha1

Описывает объект машины для Cluster API Provider Static.

- **apiVersion**
строка

Версия API объекта.

- **kind**
строка

Тип объекта.

- **metadata**
объект

- **spec**
объект

Желаемое состояние объекта StaticInstance.

- **spec.address**
строка

Обязательный параметр

IP-адрес сервера (виртуальной машины) для подключения.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **spec.credentialsRef**
объект

Обязательный параметр

Ссылка на ресурс SSHCredentials.

- **spec.credentialsRef.apiVersion**
строка

Версия API ресурса.

- **spec.credentialsRef.fieldPath**
строка

Если речь идет о части объекта, а не о целом объекте, эта строка должна содержать действительное выражение для доступа к полю JSON/Go, например `desiredState.manifest.containers[2]`.

Например, если ссылка на объект относится к контейнеру внутри пода, этот параметр будет иметь значение вида: `spec.containers{name}` (где `name` — имя контейнера, который вызвал событие) или, если имя контейнера не указано, — `spec.containers[2]` (контейнер с индексом 2 в этом поде). Этот синтаксис выбран только для того, чтобы иметь четко определенный способ ссылки на часть объекта.

- **spec.credentialsRef.kind**
строка

Kind ресурса.

- **spec.credentialsRef.name**
строка

..

Имя ресурса.

- **spec.credentialsRef.namespace**

строка

Неймспейс ресурса.

Подробнее: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **spec.credentialsRef.resourceVersion**

строка

Конкретная версия ресурса, на которую ссылается данная ссылка, если таковая имеется.

Подробнее: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **spec.credentialsRef.uid**

строка

UID ресурса.

Подробнее: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

- **status**

объект

StaticInstanceStatus defines the observed state of StaticInstance.

- **status.conditions**

массив объектов

Conditions defines current service state of the StaticInstance.

Condition contains details for one aspect of the current state of this API Resource.

- **status.conditions.lastTransitionTime**

строка

Обязательный параметр

lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**

строка

Обязательный параметр

message is a human readable message indicating details about the transition. This may be an empty string.

Максимальная длина: 32768

- **status.conditions.observedGeneration**

целочисленный

observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

Допустимые значения: 0 <= X

- **status.conditions.reason**

строка

Обязательный параметр

reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

Шаблон: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Длина: 1..1024

- **status.conditions.status**

строка

Обязательный параметр

status of the condition, one of True, False, Unknown.

Допустимые значения: True, False, Unknown

- **status.conditions.type**

строка

Обязательный параметр

type of condition in CamelCase or in foo.example.com/CamelCase.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/*)?([A-Za-z0-9](-A-Za-z0-9_\.)*)?[A-Za-z0-9]$`

Максимальная длина: 316

- **status.currentStatus**

объект

- **status.currentStatus.lastUpdateTime**

строка

- **status.currentStatus.phase**

строка

Допустимые значения: Error, Pending, Bootstrapping, Running, Cleaning

- **status.machineRef**

объект

The reference to the StaticMachine object.

- **status.machineRef.apiVersion**

строка

API version of the referent.

- **status.machineRef.fieldPath**

строка

If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as `desiredState.manifest.containers[2]`. For example, if the object reference is to a container within a pod, this would take on a value like: `"spec.containers{name}"` (where "name" refers to the name of the container that triggered the event) or if no container name is specified `"spec.containers[2]"` (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **status.machineRef.kind**

строка

Kind of the referent. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **status.machineRef.name**

строка

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **status.machineRef.namespace**

строка

Namespace of the referent. More info:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **status.machineRef.resourceVersion**

строка

Specific resourceVersion to which this reference is made, if any. More info:

<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **status.machineRef.uid**

строка

UID of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

- **status.nodeRef**

объект

The reference to the `Node` object.

- **status.nodeRef.apiVersion**

строка

API version of the referent.

- **status.nodeRef.fieldPath**

строка

If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as `desiredState.manifest.containers[2]`. For example, if the object reference is to a container within a pod, this would take on a value like: `"spec.containers{name}"` (where "name" refers to the name of the container that triggered the event) or if no container name is specified `"spec.containers[2]"` (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **status.nodeRef.kind**

строка

Kind of the referent. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **status.nodeRef.name**

строка

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **status.nodeRef.namespace**

строка

Namespace of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **status.nodeRef.resourceVersion**

строка

Specific resourceVersion to which this reference is made, if any. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **status.nodeRef.uid**

строка

UID of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

Описывает объект машины для Cluster API Provider Static.

- **apiVersion**

строка

Версия API объекта.

- **kind**

строка

Тип объекта.

- **metadata**

объект

- **spec**

объект

Желаемое состояние объекта StaticInstance.

- **spec.address**

строка

Обязательный параметр

IP-адрес сервера (виртуальной машины) для подключения.

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **spec.credentialsRef**

объект

Обязательный параметр

Ссылка на ресурс SSHCredentials.

- **spec.credentialsRef.apiVersion**

строка

Версия API ресурса.

- **spec.credentialsRef.fieldPath**

строка

Если речь идет о части объекта, а не о целом объекте, эта строка должна содержать действительное выражение для доступа к полю JSON/Go, например `desiredState.manifest.containers[2]`.

Например, если ссылка на объект относится к контейнеру внутри пода, этот параметр будет иметь значение вида: `spec.containers{name}` (где `name` — имя контейнера, который вызвал событие) или, если имя контейнера не указано, — `spec.containers[2]` (контейнер с индексом 2 в этом поде). Этот синтаксис выбран только для того, чтобы иметь четко определенный способ ссылки на часть объекта.

- **spec.credentialsRef.kind**

строка

Kind ресурса.

- **spec.credentialsRef.name**

строка

Имя ресурса.

- **spec.credentialsRef.namespace**

строка

Неймспейс ресурса.

Подробнее: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **spec.credentialsRef.resourceVersion**

строка

Конкретная версия ресурса, на которую ссылается данная ссылка, если таковая имеется.

Подробнее: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **spec.credentialsRef.uid**

строка

UID ресурса.

Подробнее: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

- **status**

объект

StaticInstanceStatus defines the observed state of StaticInstance.

- **status.conditions**

массив объектов

Conditions defines current service state of the StaticInstance.

Condition contains details for one aspect of the current state of this API Resource.

- **status.conditions.lastTransitionTime**

строка

Обязательный параметр

lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**

строка

Обязательный параметр

message is a human readable message indicating details about the transition. This may be an empty string.

Максимальная длина: 32768

- **status.conditions.observedGeneration**

целочисленный

observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

Допустимые значения: 0 <= X

- **status.conditions.reason**

строка

Обязательный параметр

reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

Шаблон: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Длина: 1..1024

- **status.conditions.status**

строка

Обязательный параметр

status of the condition, one of True, False, Unknown.

Допустимые значения: True, False, Unknown

- **status.conditions.type**

строка

Обязательный параметр

type of condition in CamelCase or in foo.example.com/CamelCase.

Шаблон: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*\/)?([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9]$`

Максимальная длина: 316

- **status.currentStatus**

объект

- **status.currentStatus.lastUpdateTime**

строка

- **status.currentStatus.phase**

строка

Допустимые значения: Error, Pending, Bootstrapping, Running, Cleaning

- **status.machineRef**

объект

The reference to the StaticMachine object.

- **status.machineRef.apiVersion**

строка

API version of the referent.

- **status.machineRef.fieldPath**

строка

If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as `desiredState.manifest.containers[2]`. For example, if the object reference is to a container within a pod, this would take on a value like: `"spec.containers{name}"` (where "name" refers to the name of the container that triggered the event) or if no container name is specified `"spec.containers[2]"` (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **status.machineRef.kind**

строка

Kind of the referent. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **status.machineRef.name**

строка

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **status.machineRef.namespace**

строка

Namespace of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **status.machineRef.resourceVersion**

строка

Specific resourceVersion to which this reference is made, if any. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **status.machineRef.uid**

строка

UID of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

- **status.nodeRef**

объект

The reference to the `Node` object.

- **status.nodeRef.apiVersion**

строка

API version of the referent.

- **status.nodeRef.fieldPath**

строка

If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as `desiredState.manifest.containers[2]`. For example, if the object reference is to a container within a pod, this would take on a value like: `"spec.containers{name}"` (where "name" refers to the name of the container that triggered the event) or if no container name is specified `"spec.containers[2]"` (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **status.nodeRef.kind**

строка

Kind of the referent. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **status.nodeRef.name**

строка

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **status.nodeRef.namespace**
строка

Namespace of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

- **status.nodeRef.resourceVersion**
строка

Specific resourceVersion to which this reference is made, if any. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency>

- **status.nodeRef.uid**
строка

UID of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids>

FAQ

Как добавить master-узлы в облачном кластере?

Порядок конвертирования кластера с одним master-узлом в мультимастерный кластер описан в FAQ модуля `control-plane-manager`.

Как уменьшить число master-узлов в облачном кластере?

Порядок конвертирования мультимастерного кластера в кластер с одним master-узлом описан в FAQ модуля `control-plane-manager`.

Статические узлы

Добавить статический узел в кластер можно вручную (пример) или с помощью Cluster API Provider Static.

Как добавить статический узел в кластер (Cluster API Provider Static)?

Чтобы добавить статический узел в кластер (сервер bare-metal или виртуальную машину), выполните следующие шаги:

1. Подготовьте необходимые ресурсы:

- Выделите сервер или виртуальную машину и убедитесь, что узел имеет необходимую сетевую связанность с кластером.
- При необходимости установите дополнительные пакеты ОС и настройте точки монтирования, которые будут использоваться на узле.

2. Создайте пользователя с правами `sudo`:

- Добавьте нового пользователя (в данном примере — `caps`) с правами выполнения команд через `sudo`:

```
useradd -m -s /bin/bash caps
usermod -aG sudo caps
```

- Разрешите пользователю выполнять команды через `sudo` без ввода пароля. Для этого отредактируйте конфигурацию `sudo` (отредактировав файл `/etc/sudoers`, выполнив команду `sudo visudo` или другим способом):

```
caps ALL=(ALL) NOPASSWD: ALL
```

3. На сервере откройте файл `/etc/ssh/sshd_config` и убедитесь, что параметр `UsePAM` установлен в значение `yes`. Затем перезапустите службу `sshd`:

```
sudo systemctl restart sshd
```

4. Сгенерируйте на сервере пару SSH-ключей с пустой парольной фразой:

```
ssh-keygen -t ed25519 -f caps-id -C "" -N ""
```

Приватный и публичный ключи будут сохранены в файлах `caps-id` и `caps-id.pub` соответственно в текущей директории.

5. Добавьте полученный публичный ключ в файл `/home/caps/.ssh/authorized_keys` пользователя `caps`, выполнив в директории с ключами на сервере следующие команды:

```
mkdir -p /home/caps/.ssh
cat caps-id.pub >> /home/caps/.ssh/authorized_keys
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps/
```

6. Создайте ресурс `SSHCredentials`.
7. Создайте ресурс `StaticInstance`.
8. Создайте ресурс `NodeGroup` с `nodeType Static`, указав желаемое количество узлов в группе и, при необходимости, фильтр выбора `StaticInstance`.

Пример добавления статического узла.

Как добавить несколько статических узлов в кластер вручную?

Используйте существующий или создайте новый кастомный ресурс (Custom Resource) `NodeGroup` (пример `NodeGroup` с именем `worker`).

Автоматизировать процесс добавления узлов можно с помощью любой платформы автоматизации. Далее приведен пример для Ansible.

1. Получите один из адресов Kubernetes API-сервера. Обратите внимание, что IP-адрес должен быть доступен с узлов, которые добавляются в кластер:

```
d8 k -n default get ep kubernetes -o json | jq '.subsets[0].addresses[0].ip + ":" + (.subsets[0].ports[0].port | tostring)' -r
```

Проверьте версию K8s. Если версия `>= 1.25`, создайте токен `node-group`:

```
d8 k create token node-group --namespace d8-cloud-instance-manager --duration 1h
```

Сохраните полученный токен, и добавьте в поле `token:` playbook'a Ansible на дальнейших шагах.

2. Если версия Kubernetes меньше 1.25, получите Kubernetes API-токен для специального `ServiceAccount`'а, которым управляет Deckhouse:

```
d8 k -n d8-cloud-instance-manager get $(d8 k -n d8-cloud-instance-manager get secret -o name | grep node-group-token) \
-o json | jq '.data.token' -r | base64 -d && echo ""
```

3. Создайте Ansible playbook с `vars`, которые заменены на полученные на предыдущих шагах значения:

```

- hosts: all
  become: yes
  gather_facts: no
  vars:
    kube_apiserver: <KUBE_APISERVER>
    token: <TOKEN>
  tasks:
    - name: Check if node is already bootstrapped
      stat:
        path: /var/lib/bashible
      register: bootstrapped
    - name: Get bootstrap secret
      uri:
        url: "https://{{ kube_apiserver }}/api/v1/namespaces/d8-cloud-instance-manager/secrets/manual-bootstrap-for-{{ node_group }}"
        return_content: yes
        method: GET
        status_code: 200
        body_format: json
        headers:
          Authorization: "Bearer {{ token }}"
        validate_certs: no
      register: bootstrap_secret
      when: bootstrapped.stat.exists == False
    - name: Run bootstrap.sh
      shell: "{{ bootstrap_secret.json.data['bootstrap.sh'] | b64decode }}"
      args:
        executable: /bin/bash
      ignore_errors: yes
      when: bootstrapped.stat.exists == False
    - name: wait
      wait_for_connection:
        delay: 30
      when: bootstrapped.stat.exists == False

```

4. Определите дополнительную переменную `node_group`. Значение переменной должно совпадать с именем `NodeGroup`, которой будет принадлежать узел. Переменную можно передать различными способами, например с использованием inventory-файла:

```

[system]
system-0
system-1

[system:vars]
node_group=system

[worker]
worker-0
worker-1

[worker:vars]
node_group=worker

```

5. Запустите выполнение playbook'a с использованием inventory-файла.

Как вручную очистить статический узел?



Инструкция справедлива как для узла, настроенного вручную (с помощью бутстрап-скрипта), так и для узла, настроенного с помощью CAPS.

Чтобы вывести из кластера узел и очистить сервер (BM), выполните следующую команду на узле:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

Можно ли удалить StaticInstance?

StaticInstance, находящийся в состоянии `Pending` можно удалять без каких-либо проблем.

Чтобы удалить StaticInstance находящийся в любом состоянии, отличном от `Pending` (`Running`, `Cleaning`, `Bootstrapping`), выполните следующие шаги:

1. Добавьте лейбл `"node.deckhouse.io/allow-bootstrap": "false"` в StaticInstance.

Пример команды для добавления лейбла:

```
d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
```

2. Дождитесь, пока StaticInstance перейдет в статус `Pending`.

Для проверки статуса StaticInstance используйте команду:

```
d8 k get staticinstances
```

3. Удалите StaticInstance.

Пример команды для удаления StaticInstance:

```
d8 k delete staticinstance d8cluster-worker
```

4. Уменьшите значение параметра `NodeGroup.spec.staticInstances.count` на 1.

Как изменить IP-адрес StaticInstance?

Изменить IP-адрес в ресурсе `StaticInstance` нельзя. Если в `StaticInstance` указан ошибочный адрес, то нужно удалить StaticInstance и создать новый.

Как мигрировать статический узел настроенный вручную под управление CAPS?

Необходимо выполнить очистку узла, затем добавить узел под управление CAPS.

Почему SSH-ключ и sudo-пароль в SSHCredentials отображаются как <omitted>?

Параметры `privateSSHKey`, `sudoPassword (v1alpha1)` и `sudoPasswordEncoded (v1alpha2)` ресурса `SSHCredentials` содержат конфиденциальные данные и защищены от просмотра.

Если у пользователя нет прав на чтение конфиденциальных данных `SSHCredentials`, API возвращает `<omitted>` вместо фактического значения. При этом остальные параметры — `user`, `sshPort`, `sshExtraArgs` — и метаданные ресурса остаются доступными пользователям, которым разрешено чтение `SSHCredentials`.

Дополнительно kube-apiserver защищает эти данные следующим образом:

- удаляет аннотацию `kubectl.kubernetes.io/last-applied-configuration` из ответов API, если она может содержать копию конфиденциальных значений;
- заменяет конфиденциальные значения на `"*****"` в событиях аудита независимо от прав пользователя и уровня аудита;
- шифрует ресурс в etcd с использованием того же механизма, что и для секретов Kubernetes, если включён параметр `apiserver.encryptionEnabled` модуля `control-plane-manager`.

Незамаскированные значения доступны контроллеру CAPS, которому они необходимы для подключения к узлу по SSH, а также пользователям с уровнем доступа `SuperAdmin` и участникам группы `kubeadm:cluster-admins`.

Роли `d8:manage:infrastructure:viewer` и `d8:manage:infrastructure:manager` позволяют читать ресурс `SSHCredentials`, но не предоставляют доступ к SSH-ключу и sudo-паролю.

Чтобы проверить, может ли пользователь читать конфиденциальные данные `SSHCredentials`, выполните:

```
d8 k auth can-i get sshcredentials/sensitive --as=<user>
```

Значение `<omitted>` означает, что фактическое значение скрыто от текущего пользователя. Это поведение применяется как к `v1alpha1`, так и к `v1alpha2`, поэтому использование другой версии API не позволяет получить скрытые данные.

Как изменить NodeGroup у статического узла?

Если узел находится под управлением CAPS, то изменить принадлежность к NodeGroup у такого узла **нельзя**. Единственный вариант — удалить StaticInstance и создать новый.

Чтобы перенести существующий статический узел созданный вручную из одной NodeGroup в другую, необходимо изменить у узла лейбл группы:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

Применение изменений потребует некоторого времени.

Как очистить узел для последующего ввода в кластер?

Это необходимо только в том случае, если нужно переместить статический узел из одного кластера в другой. Имейте в виду, что эти операции удаляют данные локального хранилища. Если необходимо просто изменить NodeGroup, следуйте этой инструкции.



Если на защищаемом узле есть пулы хранения LINSTOR/DRBD, то предварительно перенесите ресурсы с узла и удалите узел LINSTOR/DRBD, следуя инструкции.

Для всех узлов кроме control-plane

1. Удалите узел из кластера Kubernetes:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

2. Запустите на узле скрипт очистки:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

3. После перезагрузки узла запустите скрипт `bootstrap.sh`.

Для узлов control-plane

1. Снимите лейблы `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master` и `node.deckhouse.io/group` с узла:

```
d8 k label node <node> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master-
node.deckhouse.io/group-
```

2. Убедитесь, что удаляемый узел с control-plane пропал из списка членов кластера etcd:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-
plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert
/etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints
https://127.0.0.1:2379/ member list -W table
```

3. Удалите узел из кластера Kubernetes:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

4. Запустите на узле скрипт очистки:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

5. После перезагрузки узла запустите скрипт `bootstrap.sh`.

6. Дождитесь прохождения очередей Deckhouse и убедитесь, что член кластера etcd снова появился в списке:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ member list -W table
```

Как понять, что что-то пошло не так?

Если узел в NodeGroup не обновляется (значение `UPTODATE` при выполнении команды `d8 k get nodegroup` меньше значения `NODES`) или вы предполагаете какие-то другие проблемы, которые могут быть связаны с модулем `node-manager`, нужно проверить логи сервиса `bashible`. Сервис `bashible` запускается на каждом узле, управляемом модулем `node-manager`.

Чтобы проверить логи сервиса `bashible`, выполните на узле следующую команду:

```
journalctl -fu bashible
```

Пример вывода, когда все необходимые действия выполнены:

```
May 25 04:39:16 kube-master-0 systemd[1]: Started Bashible service.
May 25 04:39:16 kube-master-0 bashible.sh[1976339]: Configuration is in sync, nothing to do.
May 25 04:39:16 kube-master-0 systemd[1]: bashible.service: Succeeded.
```

Как посмотреть, что в данный момент выполняется на узле при его создании?

Если необходимо узнать, что происходит на узле (например, узел долго создается), можно проверить логи `cloud-init`. Для этого выполните следующие шаги:

1. Найдите узел, который находится в стадии бутстрапа:

```
d8 k get instances | grep Pending
```

Пример:

```
d8 k get instances | grep Pending
dev-worker-2a6158ff-6764d-nrtbj Pending 46s
```

2. Получите информацию о параметрах подключения для просмотра логов:

```
d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
```

Пример:

```
d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```


3. Выполните полученную команду (в примере выше — `curl -N http://192.168.199.158:8000`), чтобы просмотреть логи `cloud-init` и определить, на каком этапе остановилась настройка узла.

Логи первоначальной настройки узла находятся в `/var/log/cloud-init-output.log`.

Как обновить ядро на узлах?

Для дистрибутивов, основанных на Debian

Создайте ресурс `NodeGroupConfiguration`, указав в переменной `desired_version` shell-скрипта (параметр `spec.content` ресурса) желаемую версию ядра:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

Для дистрибутивов, основанных на CentOS

Создайте ресурс `NodeGroupConfiguration`, указав в переменной `desired_version` shell-скрипта (параметр `spec.content` ресурса) желаемую версию ядра:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
weight: 32
content: |
  # Copyright 2022 Flant JSC
  #
  # Licensed under the Apache License, Version 2.0 (the "License");
  # you may not use this file except in compliance with the License.
  # You may obtain a copy of the License at
  #
  #     http://www.apache.org/licenses/LICENSE-2.0
  #
  # Unless required by applicable law or agreed to in writing, software
  # distributed under the License is distributed on an "AS IS" BASIS,
  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  # See the License for the specific language governing permissions and
  # limitations under the License.

  desired_version="3.10.0-1160.42.2.el7.x86_64"

  bb-event-on 'bb-package-installed' 'post-install'
  post-install() {
    bb-log-info "Setting reboot flag due to kernel was updated"
    bb-flag-set reboot
  }

  version_in_use="$(uname -r)"

  if [[ "$version_in_use" == "$desired_version" ]]; then
    exit 0
  fi

  bb-deckhouse-get-disruptive-update-approval
  bb-dnf-install "kernel-${desired_version}"

```

На что влияют параметры NodeGroup?

Параметр NG	Disruption update	Перезаказ узлов	Рестарт kubelet
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+
kubernetesVersion	-	-	+

Параметр NG	Disruption update	Перезаказ узлов	Рестарт kubelet
nodeTemplate	-	-	-
static	-	-	+
update.maxConcurrent	-	-	-

Подробнее о всех параметрах можно прочитать в описании кастомного ресурса `NodeGroup`.

При изменении параметров `InstanceClass` или `instancePrefix` будет происходить процесс, аналогичный обновлению `Deployment` в Kubernetes (при изменении `InstanceClass` будет создан новый `MachineSet`, а при изменении `instancePrefix` — новый `MachineDeployment`). Новые инстансы будут создаваться в соответствии со значением параметра `cloudInstances.maxSurgePerZone`. По мере их создания старые инстансы будут постепенно удаляться с предварительным выполнением операции `drain`.

При обновлении, которое требует прерывания работы узла (`disruption update`), выполняется процесс вытеснения подов с узла. Если какой-либо под не может быть вытеснен, попытка повторяется каждые 20 секунд до достижения глобального таймаута в 5 минут. После истечения этого времени, поды, которые не удалось вытеснить, удаляются принудительно.

Как пересоздать эфемерные машины в облаке с новой конфигурацией?

При изменении конфигурации Deckhouse (как в модуле `node-manager`, так и в любом из облачных провайдеров) виртуальные машины не будут перезаказаны. Пересоздание происходит только после изменения ресурсов `InstanceClass` или `NodeGroup`.

Чтобы принудительно пересоздать все узлы, связанные с ресурсом `Machines`, следует добавить/изменить аннотацию `manual-rollback-id` в `NodeGroup`: `d8 k annotate NodeGroup имя_ng "manual-rollback-id=$(uuidgen)" --overwrite`.

Как выделить узлы под специфические нагрузки?



Запрещено использование домена `deckhouse.io` в ключах `labels` и `taints` у `NodeGroup`. Он зарезервирован для компонентов Deckhouse. Следует отдавать предпочтение в пользу ключей `dedicated` или `dedicated.client.com`.

Для решений данной задачи существуют два механизма:

1. Установка лейблов в `NodeGroup` `spec.nodeTemplate.labels` для последующего использования их в `Pod` `spec.nodeSelector` или `spec.affinity.nodeAffinity`. Указывает, какие именно узлы будут выбраны планировщиком для запуска целевого приложения.
2. Установка ограничений в `NodeGroup` `spec.nodeTemplate.taints` с дальнейшим снятием их в `Pod` `spec.tolerations`. Запрещает исполнение не разрешенных явно приложений на этих узлах.



Deckhouse по умолчанию поддерживает использование `taint`'а с ключом `dedicated`, поэтому рекомендуется применять этот ключ с любым значением для `taints` на ваших выделенных узлах.

Если требуется использовать другие ключи для `taints` (например, `dedicated.client.com`), необходимо добавить соответствующее значение ключа в массив `.spec.settings.modules.placement.customTolerationKeys`. Это обеспечит разрешение системным компонентам, таким как `cni-flannel`, использовать эти узлы.

Подробнее в статье на Habr.

Как выделить узлы под системные компоненты?

Фронтенд

Для Ingress-контроллеров используйте `NodeGroup` со следующей конфигурацией:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/frontend: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

Системные

Для компонентов подсистем Deckhouse параметр `NodeGroup` будет настроен с параметрами:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/system: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: system
```

Как ускорить заказ узлов в облаке при горизонтальном масштабировании приложений?

Самое действенное — держать в кластере некоторое количество предварительно подготовленных узлов, которые позволят новым репликам ваших приложений запускаться мгновенно. Очевидным минусом данного решения будут дополнительные расходы на содержание этих узлов.

Необходимые настройки целевой `NodeGroup` будут следующие:

1. Указать абсолютное количество предварительно подготовленных узлов (или процент от максимального количества узлов в этой группе) в параметре `cloudInstances.standby`.
2. При наличии на узлах дополнительных служебных компонентов, не обслуживаемых Deckhouse (например, `DaemonSet filebeat`), задать их процентное потребление ресурсов узла можно в параметре `standbyHolder.overprovisioningRate`.
3. Для работы этой функции требуется, чтобы как минимум один узел из группы уже был запущен в кластере. Иными словами, либо должна быть доступна одна реплика приложения, либо количество узлов для этой группы `cloudInstances.minPerZone` должно быть 1.

Пример:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

Как выключить machine-controller-manager/CAPi в случае выполнения потенциально деструктивных изменений в кластере?



Использовать эту настройку допустимо только тогда, когда вы четко понимаете, зачем это необходимо.

Для того чтобы временно отключить machine-controller-manager (MCM) и предотвратить его автоматические действия, которые могут повлиять на инфраструктуру кластера (например, удаление или пересоздание узлов), установите следующий параметр в конфигурации:

```
mcmEmergencyBrake: true
```

Для отключения CAPi установите следующий параметр в конфигурации:

```
capiEmergencyBrake: true
```

Как восстановить master-узел, если kubelet не может загрузить компоненты control plane?

Подобная ситуация может возникнуть, если в кластере с одним master-узлом на нем были удалены образы компонентов control plane (например, удалена директория `/var/lib/containerd`). В этом случае kubelet при рестарте не сможет скачать образы компонентов control plane, поскольку на master-узле нет параметров авторизации в `registry.deckhouse.ru`.

Далее приведена инструкция по восстановлению master-узла.

containerd

Для восстановления работоспособности master-узла нужно в любом рабочем кластере под управлением Deckhouse выполнить команду:

```
d8 k -n d8-system get secrets deckhouse-registry -O json |
jq -r '.data[".dockerconfigjson"]' | base64 -d |
jq -r '.auths["registry.deckhouse.ru"].auth'
```

Вывод команды нужно скопировать и присвоить переменной `AUTH` на поврежденном master-узле.

Далее на поврежденном master-узле нужно загрузить образы компонентов control-plane:

```
for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
  crictl pull --auth $AUTH $image
done
```

После загрузки образов необходимо перезапустить `kubelet`.

Как изменить CRI для NodeGroup?



Смена CRI возможна только между `Containerd` на `NotManaged` и обратно (параметр `cri.type`).

Для изменения CRI для NodeGroup, установите параметр `cri.type` в `Containerd` или в `NotManaged`.

Пример YAML-манифеста NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

Также эту операцию можно выполнить с помощью патча:

- Для `Containerd`:

```
d8 k patch nodegroup <имя NodeGroup> --type merge -p '{"spec":{"cri":
{"type":"Containerd"}}}'
```

- Для `NotManaged`:

```
d8 k patch nodegroup <имя NodeGroup> --type merge -p '{"spec":{"cri":
{"type":"NotManaged"}}}'
```



При изменении `cri.type` для NodeGroup, созданных с помощью `dhctl`, необходимо обновить это значение в `dhctl config edit provider-cluster-configuration` и настройках объекта NodeGroup.

После изменения CRI для NodeGroup модуль `node-manager` будет поочередно перезагружать узлы, применяя новый CRI. Обновление узла сопровождается простоем (disruption). В зависимости от настройки `disruption` для NodeGroup, модуль `node-manager` либо автоматически выполнит обновление узлов, либо потребует подтверждения вручную.

Почему изменение CRI могло не примениться?

При попытке сменить CRI изменения могут не вступить в силу. Наиболее частая причина — наличие на узлах лейблов `node.deckhouse.io/containerd-v2-unsupported` и `node.deckhouse.io/containerd-config=custom`.

Лейбл `node.deckhouse.io/containerd-v2-unsupported` добавляется на узел, если выполняется хотя бы одно из следующих условий:

- Версия ядра ниже 5.8;
- Версия ядра Linux входит в диапазоны 6.12.0–6.12.28 или 6.14.0–6.14.6 (эти версии подвержены уязвимости CVE-2025-37999 в EROFS);
- Версия `systemd` ниже 244;
- Отсутствует `cgroup v2`;
- Недоступна файловая система EROFS.

Лейбл `node.deckhouse.io/containerd-config=custom` выставляется, если на узле присутствуют файлы с расширением `.toml` в директориях `/etc/containerd/conf.d/` (если на узлах кластера используется CRI `containerd v1`) или `/etc/containerd/conf2.d/` (если на узлах кластера используется CRI `containerd v2`). В этом случае следует удалить такие файлы (если это не повлечёт критичных последствий для работы контейнеров) и удалить соответствующие ресурсы NodeGroupConfiguration (NGC), с помощью которых они могли быть добавлены.

Если используется Deckhouse Virtualization Platform, причиной невозможности смены CRI может быть NGC `containerd-dvcr-config.sh`. Если платформа виртуализации уже установлена и работает, этот NGC можно удалить.

Если нет возможности удалить ресурс NodeGroupConfiguration, вносящий изменения в конфигурацию `containerd` и несовместимый с версией `containerd v2`, используйте универсальный шаблон:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
spec:
  bundles:
    - '*'
  content: |
    {{- if eq .cri "ContainerdV2" }}
    # <Скрипт для изменения конфигурации для ContainerdV2>.
    {{- else }}
    # <Скрипт для изменения конфигурации для ContainerdV1>.
    {{- end }}
  nodeGroups:
    - '*'
  weight: 31
```

Кроме того, для смены CRI может понадобиться снять пользовательский лейбл `node.deckhouse.io/containerd-config=custom`. Сделать это можно с помощью команды:

```
for node in $(d8 k get nodes -l node-role.kubernetes.io/<Название NodeGroup, где меняется CRI>=); do
d8 k label $node node.deckhouse.io/containerd-config-; done
```

Как изменить CRI для всего кластера?



Смена CRI возможна только между `Containerd` на `NotManaged` и обратно (параметр `cri.type`).

Для изменения CRI для всего кластера, необходимо с помощью утилиты `dhctl` отредактировать параметр `defaultCRI` в конфигурационном файле `cluster-configuration`.

Также возможно выполнить эту операцию с помощью `d8 k patch`.

- Для `Containerd`:

```
data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r
'.data."cluster-configuration.yaml"' | base64 -d | sed "s/NotManaged/Containerd/"
| base64 -w0)"
d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-
configuration.yaml\":\"$data\"}}"
```

- Для `NotManaged`:

```
data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r
'.data."cluster-configuration.yaml"' | base64 -d | sed "s/Containerd/NotManaged/"
| base64 -w0)"
d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-
configuration.yaml\":\"$data\"}}"
```

Если необходимо, чтобы отдельные `NodeGroup` использовали другой CRI, перед изменением `defaultCRI` необходимо установить CRI для этой `NodeGroup`, как описано в документации.



Изменение `defaultCRI` влечет за собой изменение CRI на всех узлах, включая master-узлы. Если master-узел один, данная операция является опасной и может привести к полной неработоспособности кластера. Рекомендуется использовать `multimaster`-конфигурацию и менять тип CRI только после этого.

При изменении CRI в кластере для master-узлов необходимо выполнить дополнительные шаги:

1. Чтобы определить, какой узел в текущий момент обновляется в master `NodeGroup`, используйте следующую команду:

```
d8 k get nodes -l node-role.kubernetes.io/control-plane="" -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") |
.metadata.name' -r
```

2. Подтвердите остановку (disruption) для master-узла, полученного на предыдущем шаге:

```
d8 k annotate node <имя master-узла> update.node.deckhouse.io/disruption-approved=
```

3. Дождитесь перехода обновленного master-узла в `Ready`. Выполните итерацию для следующего master-узла.

Как добавить шаг для конфигурации узлов?

Дополнительные шаги для конфигурации узлов задаются с помощью кастомного ресурса `NodeGroupConfiguration`.

Как автоматически проставить на узел кастомные лейблы?

1. На узле создайте каталог `/var/lib/node_labels`.
2. Создайте в нём файл или файлы, содержащие необходимые лейблы. Количество файлов может быть любым, как и вложенность подкаталогов, их содержащих.
3. Добавьте в файлы нужные лейблы в формате `key=value`. Например:

```
example-label=test
```

4. Сохраните файлы.

При добавлении узла в кластер указанные в файлах лейблы будут автоматически проставлены на узел.



Обратите внимание, что добавить таким образом лейблы, используемые в DKP, невозможно. Работать такой метод будет только с кастомными лейблами, не пересекающимися с зарезервированными для Deckhouse.

Как развернуть кастомный конфигурационный файл containerd?



Пример `NodeGroupConfiguration` основан на функциях, заложенных в скрипте `032_configure_containerd.sh`.



Добавление кастомных настроек вызывает перезапуск сервиса `containerd`.

Bashible на узлах объединяет конфигурацию `containerd` для DKP с конфигурацией из файла:

- `/etc/containerd/conf.d/*.toml` — если в качестве CRI на узлах кластера используется `containerd v1`;
- `/etc/containerd/conf2.d/*.toml` — если в качестве CRI на узлах кластера используется `containerd v2`.



Вы можете переопределять значения параметров, которые заданы в файле `/etc/containerd/deckhouse.toml`, но их работу придётся обеспечивать самостоятельно. Также, лучше изменением конфигурации не затрагивать `master`-узлы (`nodeGroup master`).

Далее приведены примеры конфигурации ресурсов `NodeGroupConfiguration`, добавляющих кастомный конфигурационный файл для соответствующей версии `containerd`.

- Для `containerd v1`
- Для `containerd v2`


```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm.options]
      BinaryName = ""
      SystemdCgroup = true
      ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31

```

Как добавить конфигурацию для дополнительного registry?

В containerd существует два способа описания конфигурации registry: **устаревший** и **актуальный**.

Для проверки наличия **устаревшего** способа конфигурации выполните на узлах кластера следующие команды:

```

cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep
'plugins."io.containerd.grpc.v1.cri".registry.configs'

# Пример вывода:
# [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
#   [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
# [plugins."io.containerd.grpc.v1.cri".registry.configs]
#   [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]

```

Для проверки наличия **актуального** способа конфигурации выполните на узлах кластера следующую команду:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'
```

```
# Пример вывода:
```

```
# config_path = "/etc/containerd/registry.d"
```

Устаревший способ



Этот формат конфигурации containerd устарел (deprecated).



Используется в containerd v1, если Deckhouse не управляется с помощью модуля registry.

Конфигурация описывается в основном конфигурационном файле containerd `/etc/containerd/config.toml`.

Пользовательская конфигурация добавляется через механизм `toml merge`. Конфигурационные файлы из директории `/etc/containerd/conf.d` объединяются с основным файлом `/etc/containerd/config.toml`. Применение `merge` происходит на этапе выполнения скрипта `032_configure_containerd.sh`, поэтому соответствующие файлы должны быть добавлены заранее.

Пример конфигурационного файла для директории `/etc/containerd/conf.d/`:

```
[plugins]
[plugins."io.containerd.grpc.v1.cri"]
[plugins."io.containerd.grpc.v1.cri".registry]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
  endpoint = ["https://${REGISTRY_URL}"]
[plugins."io.containerd.grpc.v1.cri".registry.configs]
[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
  auth = "${BASE_64_AUTH}"
  username = "${USERNAME}"
  password = "${PASSWORD}"
[plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
  ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
  insecure_skip_verify = true
```



Добавление кастомных настроек через механизм `toml merge` вызывает перезапуск сервиса containerd.

Как добавить авторизацию в дополнительный registry (устаревший способ)?

Пример добавления авторизации в дополнительный registry при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
      username = "username"
      password = "password"
      # OR
      auth = "dXNlcml5hbWU6cGFzc3dvcmQ="
    EOF

```

Как настроить сертификат для дополнительного registry (устаревший способ)?

Пример настройки сертификата для дополнительного registry при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"

    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
      ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF

```

i Помимо containerd, сертификат можно добавить в операционную систему.

Как добавить TLS skip verify (устаревший способ)?

Пример добавления TLS skip verify при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'.
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
      insecure_skip_verify = true
    EOF

```

После применения конфигурационного файла проверьте доступ к registry с узлов, используя команду:

```

# Через cri-интерфейс.
crictl pull private.registry.example/image/repo:tag

```

Как настроить зеркало для доступа к публичным registries (устаревший способ)?

Пример настройки зеркала к публичным registries при использовании **устаревшего** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
mkdir -p /etc/containerd/conf.d
bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
[plugins]
[plugins."io.containerd.grpc.v1.cri"]
[plugins."io.containerd.grpc.v1.cri".registry]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
  endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
  endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]

EOF

```

Новый способ



Используется в containerd v2.

Используется в containerd v1, если управление осуществляется через модуль `registry` (например, в режиме `Direct`).

Конфигурация описывается в каталоге `/etc/containerd/registry.d` и задаётся через создание подкаталогов с именами, соответствующими адресу registry:

```

/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.ru
    ├── ca.crt
    └── hosts.toml

```

Пример содержимого файла `hosts.toml`:

```
[host]
# Mirror 1.
[host."https://${REGISTRY_URL_1}"]
  capabilities = ["pull", "resolve"]
  ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

[host."https://${REGISTRY_URL_1}.auth"]
  username = "${USERNAME}"
  password = "${PASSWORD}"

# Mirror 2.
[host."http://${REGISTRY_URL_2}"]
  capabilities = ["pull", "resolve"]
  skip_verify = true
```

i Изменения конфигураций не приводят к перезапуску сервиса containerd.

Как добавить авторизацию в дополнительный registry (актуальный способ)?

Пример добавления авторизации в дополнительный registry при использовании **актуального** способа конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Шаг может быть любой, т.к. не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
      [host."https://${REGISTRY_URL}.auth"]
        username = "username"
        password = "password"
    EOF
```


Как настроить сертификат для дополнительного registry (актуальный способ)?

Пример настройки сертификата для дополнительного registry при использовании **актуального** способа конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-tls.sh
spec:
  # Шаг может быть любой, тк не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        ca = ["/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt"]
    EOF
```



Помимо containerd, сертификат можно добавить в операционную систему.

Как добавить TLS skip verify (актуальный способ)?

Пример добавления TLS skip verify при использовании **актуального** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-skip-tls.sh
spec:
  # Шаг может быть любой, тк не требуется перезапуск сервиса containerd.
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - << EOF
    [host]
      [host."https://${REGISTRY_URL}"]
        capabilities = ["pull", "resolve"]
        skip_verify = true
    EOF

```

После применения конфигурационного файла проверьте доступ к registry с узлов, используя команды:

```

# Через cri интерфейс.
crictl pull private.registry.example/image/repo:tag

# Через ctr с указанием директории с конфигурациями.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/image/repo:tag

# Через ctr для http репозитория.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/image/repo:tag

```

Как настроить зеркало для доступа к публичным registries (актуальный способ)?

Пример настройки зеркала к публичным registries при использовании **актуального** способа конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: mirror-to-harbor.sh
spec:
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF

```

Как использовать NodeGroup с приоритетом?

С помощью параметра `priority` кастомного ресурса `NodeGroup` можно задавать порядок заказа узлов в кластере. Например, можно сделать так, чтобы сначала заказывались узлы типа *spot-node*, а если они закончились — обычные узлы. Или чтобы при наличии ресурсов в облаке заказывались узлы большего размера, а при их исчерпании — узлы меньшего размера.

Пример создания двух `NodeGroup` с использованием узлов типа `spot-node`:

```

---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral

```

В приведенном выше примере, `cluster-autoscaler` сначала попытается заказать узел типа `spot-node`. Если в течение 15 минут его не получится добавить в кластер, `NodeGroup worker-spot` будет поставлен на паузу (на 20 минут) и `cluster-autoscaler` начнет заказывать узлы из `NodeGroup worker`. Если через 30 минут в кластере возникнет необходимость развернуть еще один узел, `cluster-autoscaler` сначала попытается заказать узел из `NodeGroup worker-spot` и только потом — из `NodeGroup worker`.

После того как `NodeGroup worker-spot` достигнет своего максимума (5 узлов в примере выше), узлы будут заказываться из `NodeGroup worker`.

Шаблоны узлов (labels/taints) для `NodeGroup worker` и `worker-spot` должны быть одинаковыми, или как минимум подходить для той нагрузки, которая запускает процесс увеличения кластера.

Как интерпретировать состояние группы узлов?

Ready — группа узлов содержит минимально необходимое число запланированных узлов с состоянием `Ready` для всех зон.

Пример 1. Группа узлов в состоянии `Ready`:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
    - status: "True"
      type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
    - status: "True"
      type: Ready

```

Пример 2. Группа узлов в состоянии `Not Ready`:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
    - status: "False"
      type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
    - status: "True"
      type: Ready

```

Updating — группа узлов содержит как минимум один узел, в котором присутствует аннотация с префиксом `update.node.deckhouse.io` (например, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval — группа узлов содержит как минимум один узел, в котором присутствует аннотация `update.node.deckhouse.io/disruption-required` и отсутствует аннотация `update.node.deckhouse.io/disruption-approved`.

Scaling — рассчитывается только для групп узлов с типом `CloudEphemeral`. Состояние `True` может быть в двух случаях:

1. Когда число узлов меньше *желаемого числа узлов в группе*, то есть когда нужно увеличить число узлов в группе.
2. Когда какой-то узел помечается к удалению или число узлов больше *желаемого числа узлов*, то есть когда нужно уменьшить число узлов в группе.

Желаемое число узлов — это сумма всех реплик, входящих в группу узлов.

Пример. Желаемое число узлов равно 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  ...
  desired: 2
  ...
```

Error — содержит последнюю ошибку, возникшую при создании узла в группе узлов.

Как заставить `werf` игнорировать состояние `Ready` в группе узлов?

`werf` проверяет состояние `Ready` у ресурсов и в случае его наличия дожидается, пока значение станет `True`.

Создание (обновление) ресурса `nodeGroup` в кластере может потребовать значительного времени на развертывание необходимого количества узлов. При развертывании такого ресурса в кластере с помощью `werf` (например, в рамках процесса CI/CD) развертывание может завершиться по превышении времени ожидания готовности ресурса. Чтобы заставить `werf` игнорировать состояние `nodeGroup`, необходимо добавить к `nodeGroup` следующие аннотации:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

Что такое ресурс `Instance`?

Ресурс `Instance` в Kubernetes представляет собой описание объекта эфемерной виртуальной машины, но без конкретной реализации. Это абстракция, которая используется для управления машинами, созданными с помощью таких инструментов, как `MachineControllerManager` или `Cluster API Provider Static`.

Объект не содержит спецификации. Статус содержит:

1. Ссылку на `InstanceClass`, если он существует для данной реализации.
2. Ссылку на объект `Node` Kubernetes.
3. Текущий статус машины.
4. Информацию о том, как проверить логи создания машины (появляется на этапе создания машины).

При создании или удалении машины создается или удаляется соответствующий объект `Instance`. Самостоятельно ресурс `Instance` создать нельзя, но можно удалить. В таком случае машина будет удалена из кластера (процесс удаления зависит от деталей реализации).

Когда требуется перезагрузка узлов?

Некоторые операции по изменению конфигурации узлов могут потребовать перезагрузки.

Перезагрузка узла может потребоваться при изменении некоторых настроек `sysctl`, например, при изменении параметра `kernel.yama.pttrace_scope` (изменяется при использовании команды `astra-pttrace-lock enable/disable` в Astra Linux).

Как включить задержку выключения или перезагрузки узла, пока на нём работают критичные поды?

i Доступно в редакции **EE**.

⚠ Для принятия решения о блокировке выключения DKP дополнительно опрашивает NodeGroup. Если текущий узел принадлежит к группе `master` и является единственным master-узлом в кластере, блокировка выключения для него применена не будет.

Чтобы включить механизм задержки перезагрузки или выключения узла (далее просто «выключение узла»), добавьте на под лейбл `pod.deckhouse.io/inhibit-node-shutdown` (для Deployment укажите лейбл в шаблоне пода).

Пример манифеста пода с лейблом `pod.deckhouse.io/inhibit-node-shutdown`:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```

⚠ Блокировка срабатывает только для подов с лейблом `pod.deckhouse.io/inhibit-node-shutdown` в фазе `Running` на узле. Поды без лейбла `kubelet` выключает по обычному сценарию `graceful shutdown`.

Поиск подов, блокирующих выключение, и диагностика на узле

Для поиска подов, из-за которых блокируется выключение узла, используйте команду:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -O wide
```

Для просмотра состояния `GracefulShutdownPostpone` на узле используйте команду:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?(@.type=="GracefulShutdownPostpone")]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\n"}{.message}{"\n"}{end}'
```

Возможные значения поля `reason`:

- `WaitingForShutdownSignal` — на узле работает механизм блокировки выключения, но блокировка ещё не началась;
- `PodsWithLabelAreRunningOnNode` — выключение заблокировано, поскольку на узле ещё работают поды с лейблом `pod.deckhouse.io/inhibit-node-shutdown`;
- `NoRunningPodsWithLabel` — блокировка снята, выключение может быть продолжено.

Как механизм fencing обрабатывает разные типы узлов?

Механизм `fencing` срабатывает, когда узел теряет связь с кластером, и защищает кластер от «зомби»-узлов с неопределённым состоянием. Чтобы кратковременные сбои не приводили к каскадному срабатыванию, механизм работает в два этапа: сначала помечает узел как подозрительный (на уровне gossip-протокола `memberlist/Lifeguard`), а затем изолирует его — выключает узел и (при необходимости) удаляет связанный объект `Node`.

Подробнее о работе `fencing-agent` и `fencing-controller` можно почитать в описании параметра

`spec.fencing.mode` ресурса NodeGroup.

Поведение при срабатывании зависит от типа узла (лейбл `node.deckhouse.io/type`).

Облачные узлы (CloudEphemeral, CloudPermanent)

При срабатывании механизма fencing в режиме `Watchdog`:

1. Fencing-agent перестаёт сбрасывать таймер watchdog (модуль ядра `softdog`). По истечении таймаута происходит kernel panic — это гарантирует, что на узле не останется работающей нагрузки.
2. Fencing-controller удаляет объект Node из кластера.
3. Cloud-provider-controller (MCM/CAPi) подхватывает удаление Node и удаляет соответствующую виртуальную машину и, при необходимости, заказывает новую. Таким образом, сбойный узел пересоздаётся автоматически.

Теоретически возможен сценарий, когда kubelet на «зависшей» ВМ восстановится раньше, чем cloud-provider удалит машину, и попытается заново зарегистрировать узел в кластере. Вероятность такого события минимальна благодаря следующим механизмам:

- kernel panic оставляет узел в нерабочем состоянии — для возврата требуется ручной перезапуск ВМ;
- автоматическая перезагрузка узла после kernel panic отключена на уровне ОС для всех узлов с включённым `fencing`;
- между kernel panic и моментом, когда машину успеет обработать cloud-provider, есть предохранительные тайминги.

Статические узлы (Static, CloudStatic)

Для статических узлов объект Node **не удаляется**, поскольку соответствующую машину нельзя автоматически пересоздать. В этом случае:

1. Fencing-agent выключает узел через kernel panic, чтобы исключить неопределённое поведение.
2. Fencing-controller только удаляет поды с узла.
3. Объект Node остаётся в кластере, а сам узел ожидает, пока оператор не восстановит его вручную.

Отключение автоматической перезагрузки после kernel panic

Во всех случаях при включённом `fencing` автоматическая перезагрузка узла после kernel panic отключается на уровне ОС (настройкой ядра). Это предотвращает возврат узла в кластер с неопределённым состоянием:

- **для облачных узлов** — до момента, когда cloud-provider удалит или пересоздаст виртуальную машину;
- **для статических узлов** — до ручного восстановления оператором.

Если вы наблюдаете сценарий, при котором узел с включённым `fencing` возвращается в кластер не так, как описано выше, обратитесь в поддержку. Это поможет уточнить цепочку проверок и защитных механизмов.

Как работать с GPU-узлами?



Управление GPU-узлами доступно только в DKP Enterprise Edition.

Порядок действий по добавлению GPU-узла в кластер

Начиная с Deckhouse 1.71, если в ресурсе NodeGroup есть секция `spec.gpu`, модуль `node-manager` **автоматически**:

- настраивает containerd с `default_runtime = "nvidia"`;
- применяет необходимые системные параметры (включая фиксы для NVIDIA Container Toolkit);
- разворачивает следующие системные компоненты:
 - NFD;
 - GFD;
 - NVIDIA Device Plugin;

- DCGM Exporter;
- MIG Manager (при необходимости).

Список поддерживаемых платформ NVIDIA Container Toolkit доступен в официальной документации.

i

Для корректной работы явно укажите стратегию разделения GPU в параметре `spec.gpu.sharing` (`Exclusive`, `TimeSlicing` или `MIG`).

Containerd на GPU-узлах настраивается автоматически. Не изменяйте его конфигурацию вручную (например, через ресурс `NodeGroupConfiguration` или конфигурацию на узле).

i

Если GPU используется в кластере из одного узла и выбрана стратегия `MIG`, после перезагрузки узла требуется дополнительная ручная процедура для повторного применения MIG-конфигурации. Подробнее в разделе «Особенности использования MIG в кластере из одного узла».

Чтобы добавить GPU-узел в кластер, выполните следующие действия:

1. Создайте ресурс `NodeGroup` для GPU-узлов.

Пример с включённым **TimeSlicing** (`partitionCount: 4`) и типичными `taints` и `labels`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu
spec:
  nodeType: CloudStatic # Также может быть Static или CloudEphemeral – в
                        # зависимости от вашей инфраструктуры.
  gpu:
    sharing: TimeSlicing
    timeSlicing:
      partitionCount: 4
  nodeTemplate:
    labels:
      node-role/gpu: ""
    taints:
      - key: node-role
        value: gpu
        effect: NoSchedule
```

i

Если вы используете собственные ключи taint, убедитесь, что они разрешены в `ModuleConfig global` в массиве `.spec.settings.modules.placement.customTolerationKeys`, чтобы рабочие нагрузки могли добавлять соответствующие `tolerations`.

Полная схема полей находится в описании кастомного ресурса `NodeGroup`.

2. Установите драйвер NVIDIA и NVIDIA Container Toolkit.

Выполняйте установку на самих узлах — вручную или с помощью ресурса `NodeGroupConfiguration`. Актуальные примеры `NodeGroupConfiguration` для группы узлов `gpu` (Ubuntu, Debian, CentOS, РЕД ОС, Astra Linux) приведены в FAQ модуля `gpu`.

3. После применения конфигурации выполните бутстрап и перезагрузите узлы, чтобы применить настройки и установить драйвера.
4. Проверьте установку на узле, используя команду:

```
nvidia-smi
```



Если GPU-узел был подготовлен заранее, поставлялся с предустановленными драйверами NVIDIA или на нём установлена ОС с графическим окружением, проверьте, что GPU не занята сторонними процессами.

В выводе `nvidia-smi` в секции `Processes` не должно быть сторонних процессов, использующих GPU. На узлах с графическим окружением это могут быть, например, процессы графической сессии или `display manager`: `Xorg`, `gnome-shell`, `gdm`, `sddm`, `lightdm` и другие. Такие процессы могут занимать память GPU и мешать корректной работе рабочих нагрузок, а также применению MIG-конфигурации.

Ожидаемый корректный вывод (пример):

```
root@k8s-dvp-w1-gpu:~# nvidia-smi
Tue Aug  5 07:08:48 2025
+-----+
+-----+
| NVIDIA-SMI 535.247.01                  Driver Version: 535.247.01   CUDA Version:
12.2    |
|-----+-----+-----+
+-----+
| GPU  Name                               Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf              Pwr:Usage/Cap |      Memory-Usage | GPU-Util
Compute M. |
|-----+-----+-----+
MIG M. |
|=====+=====+=====+
=====|
|   0   Tesla V100-PCIE-32GB               Off  | 00000000:65:00:0  Off  |
0 |
| N/A   32C    P0                      35W / 250W |      0MiB / 32768MiB |      0%
Default |
|-----+-----+-----+
| N/A   |
+-----+-----+-----+
+-----+
+-----+
+-----+
| Processes:
|
|   No running processes found
|
+-----+-----+-----+
+-----+
```

5. Проверьте инфраструктурные компоненты в кластере.

Поды NVIDIA в неймспейсе `d8-nvidia-gpu`:

```
d8 k -n d8-nvidia-gpu get pod
```

Ожидаемый корректный вывод (пример):

NAME	READY	STATUS	RESTARTS	AGE
gpu-feature-discovery-80ceb7d-r842q	2/2	Running	0	2m53s
nvidia-dcgm-exporter-w9v9h	1/1	Running	0	2m53s
nvidia-dcgm-njqqb	1/1	Running	0	2m53s
nvidia-device-plugin-80ceb7d-8xt8g	2/2	Running	0	2m53s

Поды NFD в неймспейсе `d8-cloud-instance-manager`:

```
d8 k -n d8-cloud-instance-manager get pods | egrep '^(NAME|node-feature-discovery)'
```

Ожидаемый корректный вывод (пример):

NAME AGE	READY	STATUS	RESTARTS
node-feature-discovery-gc-6d845765df-45vpj 3m6s	1/1	Running	0
node-feature-discovery-master-74696fd9d5-wkjk4 3m6s	1/1	Running	0
node-feature-discovery-worker-5f4kv 3m8s	1/1	Running	0

Публикация ресурсов на узле:

```
d8 k describe node <имя-узла>
```

Фрагмент вывода (пример):

```
Capacity:
  cpu:          40
  memory:       263566308Ki
  nvidia.com/gpu: 4
Allocatable:
  cpu:          39930m
  memory:       262648294441
  nvidia.com/gpu: 4
```

6. Запустите функциональные тесты.

Вариант А. Вызов `nvidia-smi` из контейнера.

1. Создайте в кластере задачу (Job):

```
apiVersion: batch/v1
kind: Job
metadata:
  name: nvidia-cuda-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
        node-role/gpu: ""
      tolerations:
        - key: "node-role"
          operator: "Equal"
          value: "gpu"
          effect: "NoSchedule"
      containers:
        - name: nvidia-cuda-test
          image: nvidia/cuda:11.6.2-base-ubuntu20.04
          imagePullPolicy: "IfNotPresent"
          command:
            - nvidia-smi
```

2. Проверьте логи командой:

```
d8 k logs job/nvidia-cuda-test
```

Пример вывода:

```
Tue Aug 5 07:48:02 2025
+-----+
+-----+
| NVIDIA-SMI 535.247.01                Driver Version: 535.247.01   CUDA
Version: 12.2                  |
+-----+-----+-----+-----+
+-----+
| GPU   Name                               Persistence-M | Bus-Id          Disp.A | Volatile
Uncorr. ECC |
| Fan   Temp   Perf              Pwr:Usage/Cap |      Memory-Usage | GPU-Util
Compute M. |
|                               |                      |
MIG M. |
|=====+=====+=====+=====+
=====|
|  0  Tesla V100-PCIE-32GB               Off | 00000000:65:00:0  Off |
0 |
| N/A    31C    P0                      23W / 250W |      0MiB / 32768MiB |      0%
Default |
|                               |                      |
N/A |
+-----+-----+-----+-----+
+-----+
+-----+
| Processes:
|
| GPU   GI    CI          PID    Type    Process name
GPU Memory |
|       ID    ID
Usage      |
|=====+=====+=====+=====+
=====|
| No running processes found
|
+-----+-----+-----+-----+
+-----+
```

Вариант Б. Пример CUDA (vectoradd).

- 1. Создайте в кластере задачу (Job):

```

apiVersion: batch/v1
kind: Job
metadata:
  name: gpu-operator-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      tolerations:
        - key: "node-role"
          operator: "Equal"
          value: "gpu"
          effect: "NoSchedule"
      containers:
        - name: gpu-operator-test
          image: nvidia/samples:vectoradd-cuda10.2
          imagePullPolicy: "IfNotPresent"

```

2. Проверьте логи командой:

```
d8 k logs job/gpu-operator-test
```

Пример вывода:

```

[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done

```

Как выполнять мониторинг GPU?

Deckhouse Kubernetes Platform автоматически устанавливает **DCGM Exporter**; метрики GPU попадают в Prometheus и доступны в Grafana.

Какие режимы работы GPU поддерживаются?

Поддерживаются следующие режимы работы GPU:

- **Exclusive** — узел публикует ресурс `nvidia.com/gpu`; каждому поду выделяется целый GPU.
- **TimeSlicing** — временное разделение одного GPU между несколькими подами (по умолчанию `partitionCount: 4`), при этом под по-прежнему запрашивает `nvidia.com/gpu`.
- **MIG (Multi-Instance GPU)** — аппаратное разделение совместимых GPU на независимые экземпляры; при профиле `all-1g.5gb` появятся ресурсы вида `nvidia.com/mig-1g.5gb`.

Примеры приведены в разделе Управление узлами: примеры.

Как посмотреть доступные MIG-профили в кластере?

Предустановленные профили находятся в ConfigMap `mig-parted-config` в пространстве имен `d8-nvidia-gpu`. Для их просмотра используйте команду:

```
d8 k -n d8-nvidia-gpu get cm mig-parted-config -o json | jq -r '.data["config.yaml"]'
```

В разделе `mig-configs` вы увидите конкретные модели ускорителей (по PCI-ID) и список совместимых MIG-профилей для каждой из них. Найдите свою видеокарту и выберите подходящий профиль — его имя указывается в `spec.gpu.mig.partedConfig` вашего `NodeGroup`. Это позволит применить правильный профиль именно к вашей карте.

Как задать свой MIG-профиль для каждой карты на узле?

Используйте значение `custom` в параметре `partedConfig` ресурса `NodeGroup` и опишите разбиение MIG по индексам GPU:

```
спес:
  gpu:
    sharing: MIG
    mig:
      partedConfig: custom
      customConfigs:
        - index: 0
          slices:
            - profile: 7g.80gb
              count: 1 # Может принимать значение от 1 до 7.
        - index: 1
          slices:
            - profile: 3g.40gb
              count: 1
            - profile: 1g.10gb
              count: 4
      # Добавьте остальные индексы по необходимости.
```

В результате модуль `node-manager` автоматически:

- добавит в `ConfigMap` конфигурацию `mig-parted-config` с именем `custom-<node-group-name>-<hash>`, где `<hash>` вычисляется на основе схемы разбиения (длинные имена групп сокращаются с добавлением хеша, чтобы имя уместилось в лейбл);
- проставит на узлах этой группы лейбл `nvidia.com/mig.config=custom-<node-group-name>-<hash>`.

Для каждой группы узлов создаётся собственная конфигурация вида `custom-<ng>-<hash>`, при этом имена конфигураций не пересекаются.

Для GPU не активируется MIG-профиль — что проверить?

1. Проверьте модель GPU. MIG поддерживается в моделях H100/A100/A30 и **не** поддерживается в моделях V100/T4. Проверить поддержку можно в таблицах профилей в руководстве NVIDIA MIG.
2. Убедитесь, что GPU не используется процессами ОС или пользовательскими приложениями. Если на узле запущено графическое окружение, `display manager` или другие процессы, использующие GPU, применение MIG-конфигурации может завершиться ошибкой или не примениться до освобождения GPU. Проверьте это командой:

```
nvidia-smi
```

3. Проверьте конфигурацию `NodeGroup`:

```
gpu:
  sharing: MIG
  mig:
    partedConfig: all-1g.5gb
```

4. Дождитесь, пока `nvidia-mig-manager` выполнит `drain` узла и переконфигурирует GPU.

Этот процесс может занять несколько минут.

Пока операция идёт, на узле стоит тaint `mig-reconfigure`. После успешного окончания тaint удаляется.

5. Ход процесса можно отслеживать по label `nvidia.com/mig.config.state` на узле: `pending`, `rebooting`, `success` (или `error`, если что-то пошло не так).
6. Если ресурсы `nvidia.com/mig-*` не появились — проверьте:

```
d8 k -n d8-nvidia-gpu logs daemonset/nvidia-mig-manager
nvidia-smi -L
```

Поддерживаются ли AMD или Intel GPU?

Сейчас Deckhouse Kubernetes Platform автоматически настраивает **только NVIDIA GPU**. Поддержка **AMD (ROCm)** и **Intel GPU** находится в проработке и планируется к добавлению в будущих релизах.

Примеры

Ниже представлены несколько примеров описания NodeGroup, а также установки плагина cert-manager для `kubectl` и задания параметра `sysctl`.

Примеры описания NodeGroup

Облачные узлы

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
  minPerZone: 1
  maxPerZone: 2
  classReference:
    kind: AWSInstanceClass
    name: test
  nodeTemplate:
    labels:
      tier: test
```

Статические узлы

Для виртуальных машин на гипервизорах или физических серверов используйте статические узлы, указав `nodeType: Static` в NodeGroup.

Пример:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Узлы в такую группу добавляются вручную с помощью подготовленных скриптов.

Также можно использовать способ добавления статических узлов с помощью Cluster API Provider Static.

Системные узлы

Ниже представлен пример манифеста группы системных узлов.

При описании NodeGroup с узлами типа Static в поле `nodeType` укажите значение `Static` и используйте поле `staticInstances` для описания параметров настройки машин статических

узлов.

При описании NodeGroup с облачными узлами типа CloudEphemeral в поле `nodeType` укажите значение `CloudEphemeral` и используйте поле `cloudInstances` для описания параметров заказа облачных виртуальных машин.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  # Пример для узлов типа Static.
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: system
  # Пример для узлов типа CloudEphemeral.
  # nodeType: CloudEphemeral
  # cloudInstances:
  #   classReference:
  #     kind: YandexInstanceClass
  #     name: large
  #   maxPerZone: 2
  #   minPerZone: 1
  #   zones:
  #     - ru-central1-d
```

Узлы с GPU

i Функциональность управления GPU-узлами доступна только в Enterprise Edition.

Для работы узлов с GPU требуются **драйвер NVIDIA** и **NVIDIA Container Toolkit**. Возможны два варианта установки драйвера:

1. **Ручная установка** — администратор устанавливает драйвер до включения узла в кластер.
2. **Автоматизация через NodeGroupConfiguration** (подробнее – в разделе Порядок действий по добавлению GPU-узла в кластер).

После того как драйвер установлен и в NodeGroup добавлен блок `spec.gpu`, `node-manager` включает полноценную поддержку GPU: автоматически разворачиваются **NFD**, **GFD**, **NVIDIA Device Plugin**, **DCGM Exporter** и, при необходимости, **MIG Manager**.

i Узлы с GPU часто помечают отдельным taint-ом (например, `node-role=gpu:NoSchedule`) — тогда по умолчанию туда не попадают обычные поды. Сервисам, которым нужен GPU, достаточно добавить `tolerations` и `nodeSelector`.

Подробная схема параметров доступна в описании кастомного ресурса NodeGroup.

Ниже представлены примеры манифестов NodeGroup для типовых режимов работы GPU (Exclusive, TimeSlicing, MIG).

Эксклюзивный режим (Exclusive)

Каждому поду выделяется целый GPU, в кластере публикуется ресурс `nvidia.com/gpu`.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu-exclusive
spec:
  nodeType: Static
  gpu:
    sharing: Exclusive
  nodeTemplate:
    labels:
      node-role/gpu: ""
    taints:
      - key: node-role
        value: gpu
        effect: NoSchedule
```

TimeSlicing (4 слота)

GPU распределяется по временным слотам: до 4 подов могут последовательно использовать одну карту. Подходит для экспериментов, CI и лёгких inference-задач.

Поды по-прежнему запрашивают ресурс `nvidia.com/gpu`.

```
spec:
  gpu:
    sharing: TimeSlicing
    timeSlicing:
      partitionCount: 4
```

MIG (профиль `all-1g.5gb`)

Физический GPU (A100, A30 и др.) делится на аппаратные экземпляры. Планировщик увидит ресурсы `nvidia.com/mig-1g.5gb`.

Полный список поддерживаемых GPU-устройств и их профили можно увидеть, воспользовавшись инструкцией.

```
spec:
  gpu:
    sharing: MIG
    mig:
      partedConfig: all-1g.5gb
```

Проверка работы: тестовая задача (CUDA vectoradd)

```

apiVersion: batch/v1
kind: Job
metadata:
  name: cuda-vectoradd
spec:
  template:
    spec:
      restartPolicy: OnFailure
      nodeSelector:
        node-role/gpu: ""
      tolerations:
        - key: node-role
          value: gpu
          effect: NoSchedule
      containers:
        - name: cuda-vectoradd
          image: nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubuntu20.04
          resources:
            limits:
              nvidia.com/gpu: 1

```

Эта задача (Job) запускает демонстрационный пример **vectoradd** из набора CUDA-samples. Если под завершается успешно (Succeeded), значит GPU-устройство доступно и корректно настроено.

Добавление статического узла в кластер

Добавление статического узла можно выполнить вручную или с помощью Cluster API Provider Static.

Вручную

Чтобы добавить новый статический узел (выделенная VM, bare-metal-сервер и т. п.) в кластер вручную, выполните следующие шаги:

1. Для CloudStatic-узлов в облачных провайдерах, перечисленных ниже, выполните описанные в документации шаги:
 - Для AWS
 - Для GCP
 - Для YC
2. Используйте существующий или создайте новый ресурс NodeGroup (пример NodeGroup с именем `worker`). Параметр `nodeType` в ресурсе NodeGroup для статических узлов должен быть `Static` или `CloudStatic`.
3. Получите код скрипта в кодировке Base64 для добавления и настройки узла.

Пример получения кода скрипта в кодировке Base64 для добавления узла в NodeGroup `worker`:

```

NODE_GROUP=worker
d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} -o json | jq
'.data."bootstrap.sh"' -r

```

4. Выполните предварительную настройку нового узла в соответствии с особенностями вашего окружения. Например:
 - добавьте необходимые точки монтирования в файл `/etc/fstab` (NFS, Ceph и т. д.);
 - установите необходимые пакеты;
 - настройте сетевую связность между новым узлом и остальными узлами кластера.
5. Зайдите на новый узел по SSH и выполните следующую команду, вставив полученную в п. 3 Base64-строку:

```
echo <Base64-КОД-СКРИПТА> | base64 -d | bash
```

С помощью Cluster API Provider Static



Если вы ранее увеличивали количество master-узлов в кластере в NodeGroup `master` (параметр `spec.staticInstances.count`), перед добавлением обычных узлов с помощью CAPS убедитесь, что не произойдет их «перехват».

Простой пример добавления статического узла в кластер с помощью Cluster API Provider Static (CAPS):

1. Подготовьте необходимые ресурсы.

- Выделите сервер (или виртуальную машину), настройте сетевую связность и т. п., при необходимости установите специфические пакеты ОС и добавьте точки монтирования которые потребуются на узле.
- Создайте пользователя (в примере — `caps`) с возможностью выполнять `sudo`, выполнив **на сервере** следующую команду:

```
useradd -m -s /bin/bash caps
usermod -aG sudo caps
```

- Разрешите пользователю выполнять команды через `sudo` без пароля. Для этого **на сервере** внесите следующую строку в конфигурацию `sudo` (отредактировав файл `/etc/sudoers`, выполнив команду `sudo visudo` или другим способом):

```
caps ALL=(ALL) NOPASSWD: ALL
```

- Сгенерируйте **на сервере** пару SSH-ключей с пустой парольной фразой:

```
ssh-keygen -t ed25519 -f caps-id -C "" -N ""
```

Публичный и приватный ключи пользователя `caps` будут сохранены в файлах `caps-id.pub` и `caps-id` в текущей директории на сервере.

- Добавьте полученный публичный ключ в файл `/home/caps/.ssh/authorized_keys` пользователя `caps`, выполнив в директории с ключами **на сервере** следующие команды:

```
mkdir -p /home/caps/.ssh
cat caps-id.pub >> /home/caps/.ssh/authorized_keys
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps/
```

В операционных системах семейства Astra Linux, при использовании модуля мандатного контроля целостности Parsec, сконфигурируйте максимальный уровень целостности для пользователя `caps`:

```
pdp1-user -i 63 caps
```

2. Создайте в кластере ресурс SSHCredentials.

В директории с ключами пользователя **на сервере** выполните следующую команду для получения закрытого ключа в формате Base64:

```
base64 -w0 caps-id
```

На любом компьютере с `kubectl`, настроенным на управление кластером, создайте переменную окружения со значением закрытого ключа созданного пользователя в Base64, полученным на предыдущем шаге:

```
CAPS_PRIVATE_KEY_BASE64=<ЗАКРЫТЫЙ_КЛЮЧ_В_BASE64>
```

Выполните следующую команду, для создания в кластере ресурса `SSHCredentials` (здесь и далее также используйте `kubectl`, настроенный на управление кластером):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: SSHCredentials
metadata:
  name: credentials
spec:
  user: caps
  privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
EOF
```

3. Создайте в кластере ресурс `StaticInstance`, указав IP-адрес сервера статического узла:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  # Укажите IP-адрес сервера статического узла.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

4. Создайте в кластере ресурс `NodeGroup`. Параметр `count` обозначает количество `staticInstances`, подпадающих под `labelSelector`, которые будут добавлены в кластер, в данном случае 1:

i Поле `labelSelector` в ресурсе `NodeGroup` является неизменным. Чтобы обновить `labelSelector`, нужно создать новую `NodeGroup` и перенести в неё статические узлы, изменив их лейблы (`labels`).

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

i Если необходимо добавить узлы в уже существующую группу узлов, укажите их желаемое количество в поле `.spec.count` `NodeGroup`.

С помощью Cluster API Provider Static для нескольких групп узлов

Пример использования фильтров в `label selector` `StaticInstance`, для группировки статических узлов и использования их в разных `NodeGroup`. В примере используются две группы узлов (`front` и `worker`), предназначенные для разных задач, которые должны содержать разные по характеристикам узлы — два сервера для группы `front` и один для группы `worker`.

1. Подготовьте необходимые ресурсы (3 сервера или виртуальные машины) и создайте ресурс `SSHCredentials`, аналогично п.1 и п.2 примера.
2. Создайте в кластере два ресурса `NodeGroup` (здесь и далее используйте `kubectl`, настроенный на управление кластером):

i Поле `labelSelector` в ресурсе `NodeGroup` является неизменным. Чтобы обновить `labelSelector`, нужно создать новую `NodeGroup` и перенести в неё статические узлы, изменив их лейблы (labels).

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: front
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

3. Создайте в кластере ресурсы `StaticInstance`, указав актуальные IP-адреса серверов:

```

d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF

```

Cluster API Provider Static: перемещение узлов между NodeGroup

В данном разделе описывается процесс перемещения статических узлов между различными NodeGroup с использованием Cluster API Provider Static (CAPS). Процесс включает изменение конфигурации NodeGroup и обновление лейблов у соответствующих StaticInstance.

Исходная конфигурация

Предположим, что в кластере уже существует NodeGroup с именем `worker`, настроенный для управления одним статическим узлом с лейблом `role: worker`.

NodeGroup `worker`:

```

apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker

```

StaticInstance static-0:

```

apiVersion: deckhouse.io/v1alpha2
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "192.168.1.100"
  credentialsRef:
    kind: SSHCredentials
    name: credentials

```

Шаги по перемещению узла между NodeGroup



В процессе переноса узлов между NodeGroup будет выполнена очистка и повторный бутстрап узла, объект Node будет пересоздан.

1. Создание новой NodeGroup для целевой группы узлов

Создайте новый ресурс NodeGroup, например, с именем front, который будет управлять статическим узлом с лейблом role: front.

```

d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: front
EOF

```

2. Обновление лейбла у StaticInstance

Измените лейбл role у существующего StaticInstance с worker на front. Это позволит новой NodeGroup front начать управлять этим узлом.

```
d8 k label staticinstance static-worker-1 role=front --overwrite
```

3. Уменьшение количества статических узлов в исходной NodeGroup

Обновите ресурс NodeGroup `worker`, уменьшив значение параметра `count` с 1 до 0.

```
d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --type=merge
```

Пример описания NodeUser

```
apiVersion: deckhouse.io/v1
kind: NodeUser
metadata:
  name: testuser
spec:
  uid: 1100
  sshPublicKeys:
    - "<SSH_PUBLIC_KEY>"
  passwordHash: <PASSWORD_HASH>
  isSudoer: true
```

Пример описания NodeGroupConfiguration

Установка плагина cert-manager для kubectl на master-узлах

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
    - "*"
  nodeGroups:
    - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-
manager/releases/download/v1.7.1/kubectl-cert_manager-linux-amd64.tar.gz -o - | tar
-zxvf - kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Задание параметра sysctl


```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
    - "*"
  nodeGroups:
    - "*"
  content: |
    sysctl -w vm.max_map_count=262144

```

Добавление корневого сертификата в хост



Данный пример приведен для ОС Ubuntu.
Способ добавления сертификатов в хранилище может отличаться в зависимости от ОС.
При адаптации скрипта под другую ОС измените параметры bundles и content.



Для использования сертификата в `containerd` (в т.ч. pull контейнеров из приватного репозитория) после добавления сертификата требуется произвести рестарт сервиса.

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
    - '*'
  bundles:
    - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAj4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    JTEjMCEGA1UEAxMabmV4dXMUNTEuMjUwLjQxLjIuc3NsXSAuaW8wHhcNMjQwODAx
    MTAzMjA4WhcNMjQxMDMwMTAzMjA4WjAlMSMwIQQYDVQQDEXpuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pbzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL1p
    WLPr2c4SZX/i4IS59Ly1USPjRE21G4pMYewUjKsXnYv7hUKHvbnL/P9dmGBm2JsL
    WFLRZbzCv7+5/J+9mPVL2TdTBWuAcTUyaG5GZ/1w64AmAWxqGMFx4eyD1zo9eSmN
    G2jis8VofL9dWdfUYhRzJ90qKxgK6k7tfhL0pv7IHDbbf28fCEnkvxsA98lGkq3H
    fUfvHV60i8pcyPZ/c8ayIf4+J0nf7oW/TgWqI7x6R1CkdzwpJ8oU7PGc0ySUWaP
    G5bH3ofBavL0bNEsyScz4TFCJ9b4a05GFA0mgjFMMUi9qXDH72sBSrgi08Dxmimg
    Hfs198SZr3br5GTJoAkCAwEAAN1MHMwDgYDVR0PAAQH/BAQDAgWgMAwGA1UdEwEB
    /wQCAAwUwYDVR0RBEEwSoIPbmV4dXMuc3ZjLmxvY2FsgphuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pb4IbZG9ja2VyLjUxLjIIMC40MS4yLnNzbGlwLmlvMA0GCsQG
    SIb3DQEBCwUAA4IBAQBvTjTTXWeWtfaUDrcp1YW1pKgZ7LTb27f3QCxukXpC+wL
    dcb4EP/vDf+UqCogKl6rCEA0i23Dtn85KAE9PQZFfI5hLuLptd0gUh03UdluoY36
    D4WvUoCfGpGx12FRdanQBBja+oDsT1Qe0pKwQJuwjpZcGfB2YZqh00UcJpC8kxtU
    by3uoxJoveHPRlBm2+ACBPPLHu/yH7st24sr1CodJHNT6P8ugIBAZxi3/Hq0wj4K
    aaQzdGXeFckWaxIny7F1M3cIWEXWzhAFnoTgrwklf7N7VWHPIvlIh1EYASsVYKn
    iATq8C7qhU0GsknDh3QSp0JeJmpcBwln11/9BGRP
    -----END CERTIFICATE-----
    EOF

```

```

)

# bb-event          - Creating subscription for event function. More
information: http://www.bashbooster.net/#event
## ca-file-updated  - Event name
## update-certs     - The function name that the event will call

bb-event-on "ca-file-updated" "update-certs"

update-certs() {          # Function with commands for adding a certificate to
the store
    update-ca-certificates
}

# bb-tmp-file - Creating temp file function. More information:
http://www.bashbooster.net/#tmp
CERT_TMP_FILE="$( bb-tmp-file )"
echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

# bb-sync-file          - File synchronization function.
More information: http://www.bashbooster.net/#sync
## "${CERTS_FOLDER}/${CERT_FILE_NAME}.cert" - Destination file
## ${CERT_TMP_FILE}      - Source file
## ca-file-updated       - Name of event that will be called
if the file changes.

bb-sync-file \
    "${CERTS_FOLDER}/${CERT_FILE_NAME}.cert" \
    ${CERT_TMP_FILE} \
    ca-file-updated

```

Добавление в containerd возможности скачивать образы из insecure container registry

Возможность скачивания образов из insecure container registry включается с помощью параметра `insecure_skip_verify` в конфигурационном файле containerd. Подробнее — в разделе Как добавить конфигурацию для дополнительного registry.

Модуль okmeter

Модуль `okmeter` устанавливает агент Okmeter как `daemonset` в пространстве имён `d8-okmeter` и удаляет ранее установленный вручную `okmeter`.

Настройки

В модуле реализован 1 алерт.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `okmeter`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot displays the Deckhouse management interface. On the left, a sidebar contains navigation options: 'Обзор' (Overview), 'Обновления' (Updates), 'Модули' (Modules), and 'Глобальные настройки' (Global settings). The 'Модули' section is active. The main panel shows a list of modules under the heading 'Модули'. The 'admission-policy-engine' module is selected, and its status is 'ВКЛЮЧЕН' (Enabled). Below the module list, there are tabs for 'Конфигурация' (Configuration), 'Ресурсы' (Resources), and 'YAML'. The 'Конфигурация' tab is active, showing a toggle switch for 'Модуль включен' (Module enabled), which is currently turned on. Other modules listed include 'backup' and 'basic-auth', both of which are 'выключен' (disabled).

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `okmeter`:

```
d8 system module enable okmeter
```

- **С помощью ModuleConfig `okmeter`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `okmeter` (создайте его, при необходимости).

Пример манифеста для включения модуля `okmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  enabled: true
```

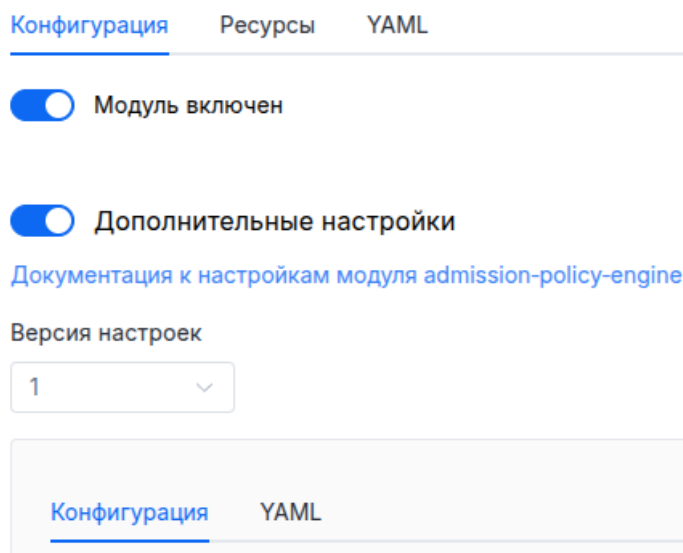
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `okmeter`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `okmeter` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `okmeter`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `okmeter` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc okmeter
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig okmeter`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig okmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.apiKey**

строка

Обязательный параметр

Ключ для доступа к API Okmeter.

Ключ можно взять в личном кабинете Okmeter (`OKMETER_API_TOKEN`).

Пример:

```
apiKey: 5ff9z2a3-9127-1sh4-2192-06a3fc6e13e3
```

- **settings.disableAutoDiscovery**

Отключает автоматическое обнаружение сервисов, которые способен мониторить okmeter. Установка опции в `true` оставит лишь базовый мониторинг хоста, прочие плагины (`postgresql`, `mysql`, `redis` и др.) будут отключены. По умолчанию установлено значение `false`.

- **settings.image**

объект

- **settings.image.repository**

строка

Обязательный параметр

Docker repository for okmeter agent image.

По умолчанию: `registry.okmeter.io/agent/okagent`

- **settings.image.tag**

строка

Обязательный параметр

Okmeter agent image tag.

По умолчанию: `latest`

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если ничего не указано или указано `false`, то Deckhouse попытается вычислить значение `nodeSelector` автоматически.

Пример:

```
nodeSelector:
  test: test
```

- **settings.tolerations**
массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если ничего не указано или указано `false`, то Deckhouse попытается вычислить значение `tolerations` автоматически.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Примеры

Пример

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  version: 1
  enabled: true
  settings:
    apiKey: 5ff9z2a3-9127-1sh4-2192-06a3fc6e13e3
```

Модуль openvpn

i Веб-интерфейсы, связанные с модулем: `openvpn-admin`

Дает доступ к ресурсам кластера через OpenVPN с аутентификацией по сертификатам, предоставляет простой веб-интерфейс.

В веб-интерфейсе можно:

- выпускать сертификаты;
- отзывать сертификаты;
- отменять отзыв сертификатов;
- получать готовый пользовательский конфигурационный файл.

Интеграция с модулем `user-authn` позволит управлять доступом пользователей в веб-интерфейс.

Варианты публикации VPN-сервиса

1. Выберите для подключения один или несколько внешних IP-адресов.
2. Воспользуйтесь одним из методов подключения:
 - по внешнему IP-адресу (`ExternalIP`) — если есть узлы с публичными IP-адресами;
 - с помощью `LoadBalancer` — для всех облачных провайдеров и их схем размещения с поддержкой заказа `LoadBalancer`;
 - `Direct` — настройте путь трафика вручную: от точки входа в кластер до пода с `OpenVPN`.

Доступные ресурсы кластера после подключения к VPN

На компьютер пользователя после подключения к VPN доставляются (push) следующие параметры:

- адрес `kube-dns` добавляется в DNS-серверы клиента для возможности прямого обращения к сервисам Kubernetes по FQDN;
- маршрут в локальную сеть;
- маршрут в сервисную сеть кластера;
- маршрут в сеть подов.

Аудит пользовательских соединений

Модуль позволяет включить логирование пользовательской активности через VPN в JSON-формате. Группировка трафика происходит по полям `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto`. С помощью модуля log-shipper логи контейнера можно собрать и отправить на хранение для последующего аудита.

Настройки

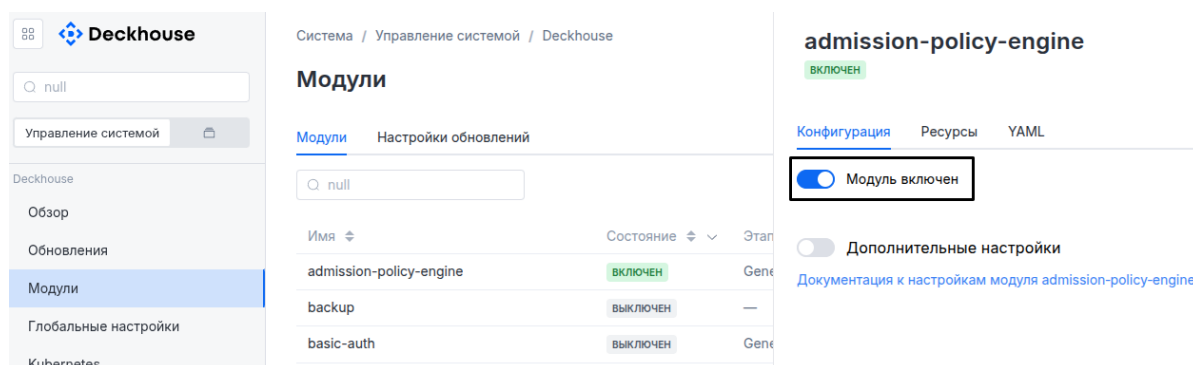
В модуле реализовано 9 алертов.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `openvpn`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `openvpn`:

```
d8 system module enable openvpn
```

- **С помощью ModuleConfig `openvpn`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `openvpn` (создайте его, при необходимости).

Пример манифеста для включения модуля `openvpn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  enabled: true
```

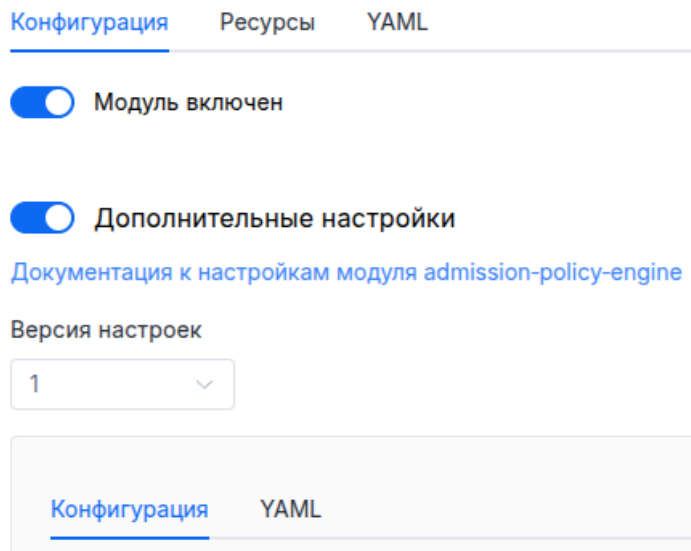
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `openvpn`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект `ModuleConfig openvpn` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `openvpn`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig openvpn` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc openvpn
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig openvpn`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig openvpn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса `ModuleConfig`, схема которого содержит номер

версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Описание конверсии отсутствует.

Преобразования, выполняемые конверсией (синтаксис jq)...

- `del(.auth.password) | if .auth == {} then del(.auth) end`
- `del(.storageClass)`

Параметры

Версия схемы: 2

- **settings**

объект

- **settings.auth**

объект

Опции, связанные с аутентификацией или авторизацией в приложении.

- **settings.auth.allowedUserEmails**

массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.allowedUserGroups**

массив строк

Массив групп пользователей с доступом в панель администратора OpenVPN.

Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.

i

Внимание. При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках `DexProvider`.

- **settings.auth.externalAuthentication**

объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX external-auth, работающий на основе модуля Nginx auth_request).

i

Внешняя аутентификация включается автоматически, если включен модуль `user-authn`.

- **settings.auth.externalAuthentication.authSignInURL**

строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

- **settings.auth.externalAuthentication.authURL**

строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

- **settings.auth.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию

для доступа в OpenVPN.

- **settings.clientCertExpirationDays**

целочисленный

Срок действия клиентского сертификата для OpenVPN в днях. Указывает количество дней, на которое будет выпущен клиентский сертификат, сгенерированный с помощью EasyRSA.

Срок действия клиентского сертификата может быть не менее 1 дня и не более 10 лет.

По умолчанию: 3650

Допустимые значения: `1 <= X <= 3650`

- **settings.externalHost**

строка

IP-адрес или домен, по которому клиенты подключаются к OpenVPN-серверу.

По умолчанию используются данные из сервиса с именем `openvpn-external`.

- **settings.externalIP**

строка

IP-адрес одного из узлов кластера, который будет использоваться для подключения клиентов OpenVPN.

Требуется только при использовании инлета `ExternalIP`.

- **settings.externalPort**

целочисленный

Порт, который вывешивается наружу на `externalIP` или балансировщике.

По умолчанию: 5416

- **settings.highAvailability**

булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

- **settings.hostPort**

целочисленный

Порт для подключения к OpenVPN-серверу, который будет доступен на узле, на котором запущен под OpenVPN.

Параметр доступен при выборе инлета `HostPort`.

По умолчанию: 5416

- **settings.https**

объект

Тип сертификата, используемого для панели администратора OpenVPN.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

- **settings.https.certManager**

объект

- **settings.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для панели администратора OpenVPN.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

- **settings.https.customCertificate**

объект

- **settings.https.customCertificate.secretName**

строка

Имя Secret'a в пространстве имен `d8-system`, который будет использоваться для панели администратора OpenVPN (данный Secret должен быть в формате `kubernetes.io/tls`).

По умолчанию: `false`

■ **settings.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — панель администратора OpenVPN будет работать по HTTPS, самостоятельно заказывая сертификат с помощью `ClusterIssuer`, заданного в параметре `certManager.clusterIssuerName`;
- `CustomCertificate` — панель администратора OpenVPN будет работать по HTTPS, используя сертификат из пространства имен `d8-system`;
- `Disabled` — панель администратора OpenVPN будет работать только по HTTP;
- `OnlyInURI` — панель администратора OpenVPN будет работать по HTTP (подразумевая, что перед ней стоит внешний HTTPS-балансировщик, который терминирует HTTPS) и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

○ **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для панели администратора OpenVPN.

По умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

○ **settings.inlet**

строка

Способы подключения из внешнего мира.

Поддерживаются следующие инлеты:

- `ExternalIP` — когда имеются узлы с публичными IP-адресами. Используется в комбинации с параметром `externalIP`;
- `LoadBalancer` — для всех облачных провайдеров и их схем размещения, в которых поддерживается заказ `LoadBalancer`'ов;
- `HostPort` — порт сервера OpenVPN будет доступен на узле, на котором находится под. За выбор порта отвечает параметр `hostPort`;
- `Direct` — для нестандартных случаев. В пространстве имен `d8-openvpn` необходимо создать сервис с именем `openvpn-external`, который отправляет трафик в под с лейблом `app: openvpn` на порт с именем `ovpn-tcp` (или просто `1194`). Из этого сервиса парсится `externalIP`, IP-адрес балансировщика или его `host`. Если ничего этого нет, необходимо указать параметр `externalHost`.

Допустимые значения: `ExternalIP`, `LoadBalancer`, `HostPort`, `Direct`

○ **settings.loadBalancer**

объект

Секция опциональных настроек для инлета `LoadBalancer`.

■ **settings.loadBalancer.annotations**

объект

Аннотации, которые будут проставлены сервису для гибкой настройки балансировщика.

i **Внимание.** Модуль не учитывает особенности указания аннотаций в различных облаках. Если аннотации для заказа балансировщика применяются только при создании сервиса, для обновления подобных параметров вам необходимо будет выключить и включить модуль.

- **settings.loadBalancer.sourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешен доступ на балансировщик.

Облачный провайдер может не поддерживать данную опцию и игнорировать ее.

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.pmacctEnabled**

булевый

Включить логирование пользовательской активности через VPN в JSON-формате.

Все соединения внутри `tun`-интерфейсов будут собраны через `libpcap` и записаны в лог в виде `{"event type": "purge", "ip src": "172.25.175.10", "ip dst": "10.222.0.10", "port src": 32172, "port_dst": 53, "ip_proto": "udp", "packets": 1, "bytes": 53}`.

По умолчанию: `false`

- **settings.pushDefaultGatewayToClient**

булевый

Направить весь трафик клиентов через VPN. У клиентов при подключении будет добавлен маршрут по умолчанию через VPN-туннель.

По умолчанию: `false`

- **settings.pushToClientDNS**

строка

Адрес DNS-сервера, который отправляется клиентам при подключении.

По умолчанию используется IP-адрес сервиса `kube-system/kube-dns`.

- **settings.pushToClientRoutes**

массив строк

Список маршрутов, которые отправляются клиентам при подключении.

По умолчанию генерируется автоматически из локальной сети кластера, сервисной подсети и подсети подов.

Пример:

```
pushToClientRoutes:
- 172.25.176.0/24
- 172.25.178.0/255.255.255.0
```

- **settings.pushToClientSearchDomains**

массив строк

Список доменов поиска, которые отправляются клиентам при подключении.

По умолчанию используется значение из переменной `global.discovery.clusterDomain`.

- **settings.tcpEnabled**

булевый

По умолчанию: `true`

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка
- **settings.tunnelNetwork**
строка

Подсеть, используемая для туннеля OpenVPN-сервера.

По умолчанию: `172.25.175.0/24`

Примеры:

```
tunnelNetwork: 172.25.175.0/24
```

```
tunnelNetwork: 172.25.178.0/255.255.255.0
```

- **settings.udpEnabled**
булевый

По умолчанию: `false`

Аутентификация

По умолчанию используется модуль user-authn. Также можно настроить аутентификацию с помощью параметра externalAuthentication. Если эти варианты отключены, модуль включит базовую аутентификацию со сгенерированным паролем.

Чтобы просмотреть сгенерированный пароль, выполните команду:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values openvpn -o json | jq '.openvpn.internal.auth.password'
```

Чтобы сгенерировать новый пароль, удалите ресурс Secret:

```
d8 k -n d8-openvpn delete secret/basic-auth
```

i Параметр `auth.password` больше не поддерживается.

FAQ

Почему не работает автоматическая настройка DNS-сервера при подключении на macOS и Linux с помощью клиента OpenVPN

В связи с архитектурными особенностями операционных систем семейства Linux и macOS автоматическая конфигурация DNS-сервера при подключении с помощью официального клиента OpenVPN невозможна.

Для настройки DNS-сервера в таких ОС сервисом предусмотрена возможность использования сторонних скриптов, которые запускаются при подключении и отключении клиента.

В клиентских конфигурациях, генерируемых модулем, предопределены и закомментированы блоки, отвечающие за эти настройки:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

Для активации указанных блоков кода необходимо их раскомментировать (удалить начальный символ `#`), а также указать корректные пути к скриптам.

Скрипты можно подготовить самостоятельно или воспользоваться готовыми решениями от официального OpenVPN Community (для Linux). Для macOS можно воспользоваться сторонним скриптом.



Скрипты должны обладать правами на исполнение.

Как отозвать, ротировать или удалить сертификат клиента

Все действия с клиентскими сертификатами выполняются через веб-интерфейс `openvpn-admin`. Справа от имени каждого пользователя доступны кнопки для управления сертификатом:

Search Table						Add user	Hide revoked
	Name	Account Status	Active Connections	Expiration Date	Revocation Date	Actions	
1	test	Active	0	2035-06-15 15:28:14		Revoke	Download config Edit routes

Чтобы ротировать (выпустить новый сертификат) или удалить клиента, необходимо сначала отозвать его текущий сертификат (Revoke):

Search Table						Add user	Hide revoked
	Name	Account Status	Active Connections	Expiration Date	Revocation Date	Actions	
1	test	Revoked	0	2035-06-15 15:28:14	2025-06-18 08:57:51	Delete	Rotate Unrevoke

После отзыва становятся доступными действия Renew (ротация) и Delete (удаление).

Как ротировать сертификат сервера

Серверный сертификат ротится автоматически за 1 день до окончания срока его действия.

Если требуется выполнить ротацию вручную (например, при повреждении сертификата или внеплановой замене), выполните следующие шаги:

1. Удалите текущий секрет, содержащий сертификат и ключ сервера:

```
d8 k -n d8-openvpn delete secrets openvpn-pki-server
```

2. Перезапустите поды OpenVPN, чтобы инициировать генерацию нового сертификата:

```
d8 k -n d8-openvpn rollout restart sts openvpn
```

Как ротировать корневой сертификат (CA)

Корневой сертификат (CA) и серверный сертификат ротится автоматически за 1 день до окончания срока действия. Автоматическая ротация сертификатов пользователя не предусмотрена. Корневой сертификат (CA) используется для подписи всех сертификатов в OpenVPN — как серверных, так и клиентских. Поэтому при его замене необходимо перевыпустить все зависимые сертификаты.

Шаги для ротации корневого сертификата:

1. Отзовите или удалите все активные клиентские сертификаты. Сделать это можно через интерфейс `openvpn-admin`. Если вы воспользуетесь отзывом, то после замены CA можно будет выполнить ротацию сертификатов (Renew), не создавая клиента заново.
2. Удалите секреты `openvpn-pki-ca` и `openvpn-pki-server` в пространстве имён `d8-openvpn`:

```
d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
```

3. Перезапустите поды OpenVPN:

```
d8 k -n d8-openvpn rollout restart sts openvpn
```

4. Выполните ротацию сертификатов для отозванных клиентов или создайте новых клиентов с новыми сертификатами.
5. Удалите все отозванные сертификаты:

```
d8 k -n d8-openvpn delete secrets -l revokedForever=true
```

Примеры

Пример для кластеров bare metal

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: ExternalIP
    externalIP: 5.4.54.4
```

Пример для AWS и Google Cloud

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

Пример для публичного IP-адреса на внешнем балансировщике

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # Внутренний IP-адрес, который примет трафик от внешнего
    балансировщика.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node

```

Модуль operator-prometheus

Модуль устанавливает Prometheus Operator, который автоматизирует развёртывание и управление инсталляциями Prometheus.

Оператор:

- Создаёт следующие кастомные ресурсы:
 - Prometheus — описывает инсталляцию (кластер) Prometheus;
 - ServiceMonitor — определяет как собирать метрики с сервисов;
 - Alertmanager — описывает кластер Alertmanager;
 - PrometheusRule — содержит набор правил Prometheus.
- Отслеживает эти ресурсы и:
 - Генерирует StatefulSet с Prometheus и необходимые конфигурационные файлы, сохраняя их в секрете (Secret).
 - При изменениях ServiceMonitor и PrometheusRule обновляет конфигурацию Prometheus путём изменения соответствующего секрета.

Prometheus

Функции Prometheus

Сервер Prometheus выполняет две основные задачи — собирает метрики и обрабатывает правила.

- Для каждой цели мониторинга (target) с указанным интервалом `scrape_interval` выполняется HTTP-запрос к этой цели. Метрики в собственном текстовом формате сохраняются в базу данных Prometheus.
- Через указанный интервал `evaluation_interval` обрабатываются правила (rules), в результате чего:
 - отправляются алерты;
 - в базу записываются новые метрики (результаты выполнения правил).

Настройка Prometheus

Конфигурация Prometheus состоит из основного файла `config` и файлов правил (`rule files`).

В `config` используются ключевые секции:

- `scrape_configs` — определяет, где и как обнаруживать цели мониторинга (targets);
 - `rule_files` — перечисляет пути к файлам правил, которые Prometheus должен загружать.
- Пример:

```

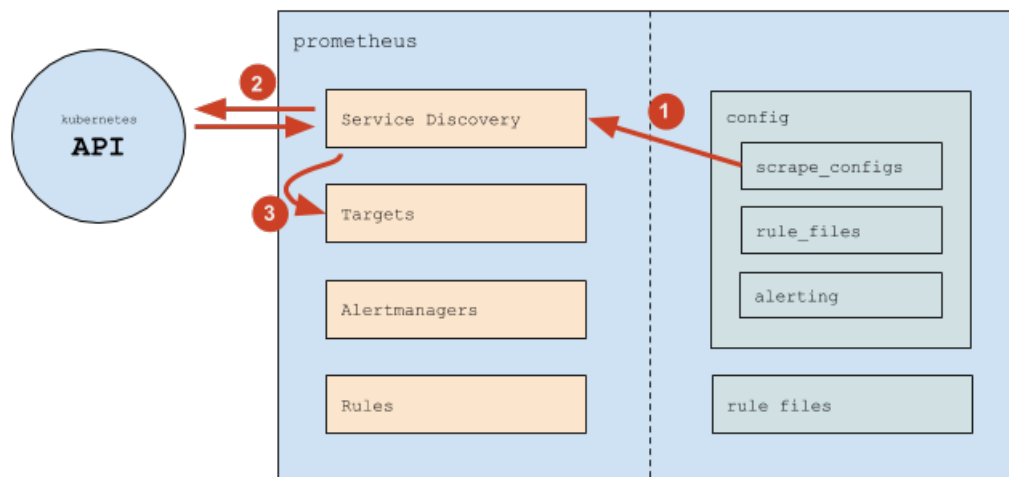
rule_files:
- /etc/prometheus/rules/rules-0/*
- /etc/prometheus/rules/rules-1/*

```

- `alerting` — задаёт параметры поиска и отправки алертов в кластеры Alertmanager. По структуре секция похожа на `scrape_configs`, но результатом будет список эндпоинтов, в которые Prometheus отправляет оповещения.

Источники целей мониторинга

Общий принцип работы Prometheus:



1. Prometheus читает секцию `scrape_configs` и настраивает внутренний механизм Service Discovery.
2. Service Discovery взаимодействует с API Kubernetes (получает эндпоинты и связанную метаинформацию).
3. На основе событий в кластере Service Discovery обновляет список целей мониторинга.

В `scrape_configs` задаётся набор `scrape job` — логических описаний того, откуда и как собирать метрики. Пример:


```

scrape_configs:
  # Общие настройки.
  - job_name: d8-monitoring/custom/0    # Название scrape job, показывается в разделе
Service Discovery.
    scrape_interval: 30s                # Как часто собирать данные.
    scrape_timeout: 10s                 # Таймаут на запрос.
    metrics_path: /metrics              # HTTP-путь.
    scheme: http                        # HTTP или HTTPS.
    # Настройки Service Discovery.
    kubernetes_sd_configs:              # Брать цели из Kubernetes.
      - api_server: null                # Адрес API-сервера используется из переменных
окружения (переменные есть в каждом поде).
        role: endpoints                # Брать цели из эндпоинтов.
        namespaces:
          names:                        # Ограничение списка пространств имён.
            - foo
            - baz
    # Настройки фильтрации (какие эндпоинты брать, а какие нет) и relabeling'a (какие
лейблы добавить или удалить, распространяется на все получаемые метрики).
    relabel_configs:
      # Фильтр по значению лейбла prometheus_custom_target (полученного из связанного с
эндпоинтом сервиса).
      - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
        regex: .+                      # Любой не пустой лейбл.
        action: keep
      # Фильтр по имени порта.
      - source_labels: [__meta_kubernetes_endpointslice_port_name]
        regex: http-metrics            # Только порт с именем http-metrics.
        action: keep
      # Добавление лейбла job. Используйте значение лейбла prometheus_custom_target у
сервиса, к которому добавлен префикс "custom-".
      #
      # Лейбл job:
      #   * определяет название группы, в которой будет показываться цель мониторинга;
      #   * добавляется к метрикам для удобной фильтрации в правилах и дашбордах.
      - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
        regex: (.*)
        target_label: job
        replacement: custom-$1
        action: replace
      # Добавление лейбла namespace.
      - source_labels: [__meta_kubernetes_namespace]
        regex: (.*)
        target_label: namespace
        replacement: $1
        action: replace
      # Добавление лейбла service.
      - source_labels: [__meta_kubernetes_service_name]
        regex: (.*)
        target_label: service
        replacement: $1
        action: replace
      # Добавление лейбла instance (в котором будет имя пода).
      - source_labels: [__meta_kubernetes_pod_name]
        regex: (.*)
        target_label: instance
        replacement: $1
        action: replace

```

Таким образом, Prometheus автоматически отслеживает добавление и удаление:

- подов (изменение эндпоинтов приводит к добавлению или удалению целей мониторинга);

- сервисов (и соответствующих эндпоинтов) в указанных пространствах имён.

Изменение конфигурации требуется, когда необходимо:

- добавить новый `scrape_config` (обычно — новый класс сервисов для мониторинга);
- изменить список пространств имён.

Prometheus Operator

Функции Prometheus Operator

Prometheus Operator с помощью механизма CRD (Custom Resource Definitions) определяет четыре кастомных ресурса:

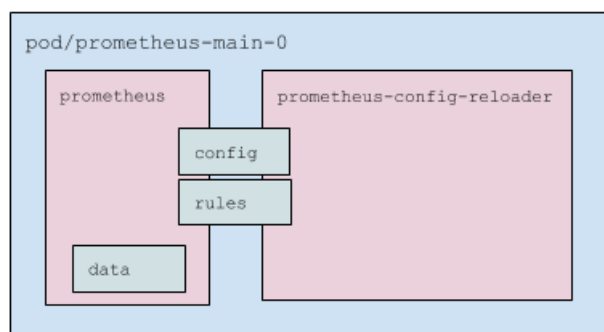
- Prometheus — описывает инсталляцию (кластер) Prometheus;
- ServiceMonitor — задаёт, как собирать метрики с набора сервисов;
- Alertmanager — описывает кластер Alertmanager;
- PrometheusRule — содержит набор правил Prometheus.

Также Prometheus Operator отслеживает ресурсы Prometheus и для каждого генерирует:

- StatefulSet с самим Prometheus;
- секрет с `prometheus.yaml` (основной конфигурационный файл) и `configmaps.json` (конфигурационный файл для `prometheus-config-reloader`).

Отслеживает ресурсы ServiceMonitor и PrometheusRule и на их основании обновляет конфигурацию (`prometheus.yaml` и `configmaps.json`) в соответствующем секрете.

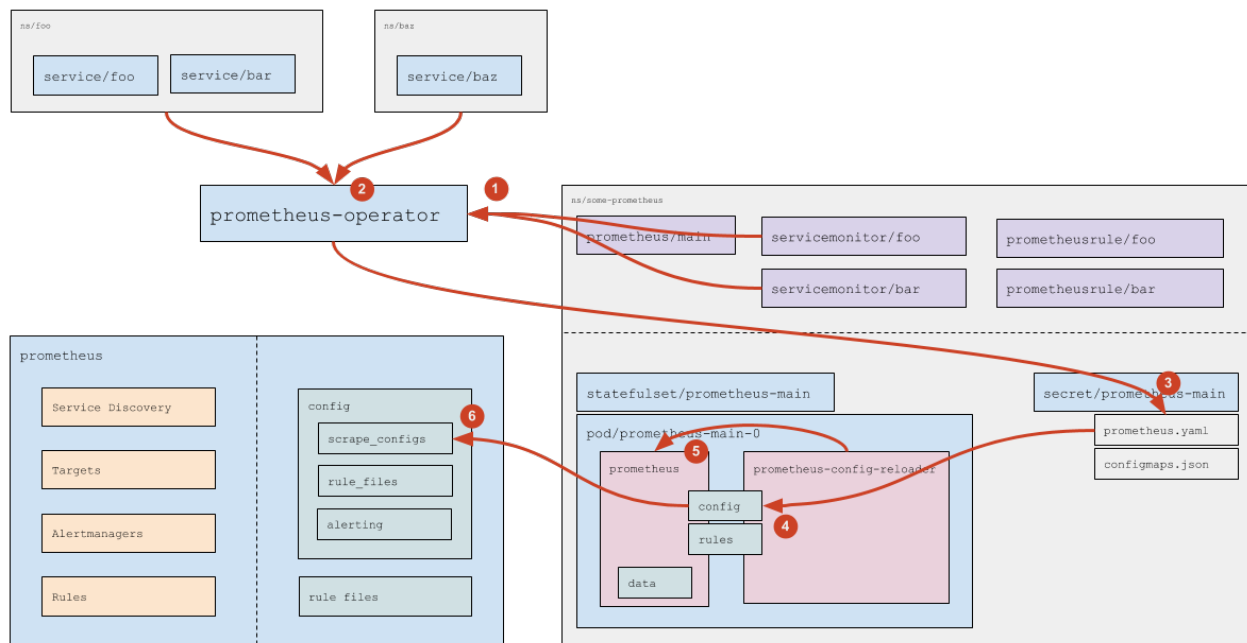
Состав пода Prometheus



Поды Prometheus представляют собой:

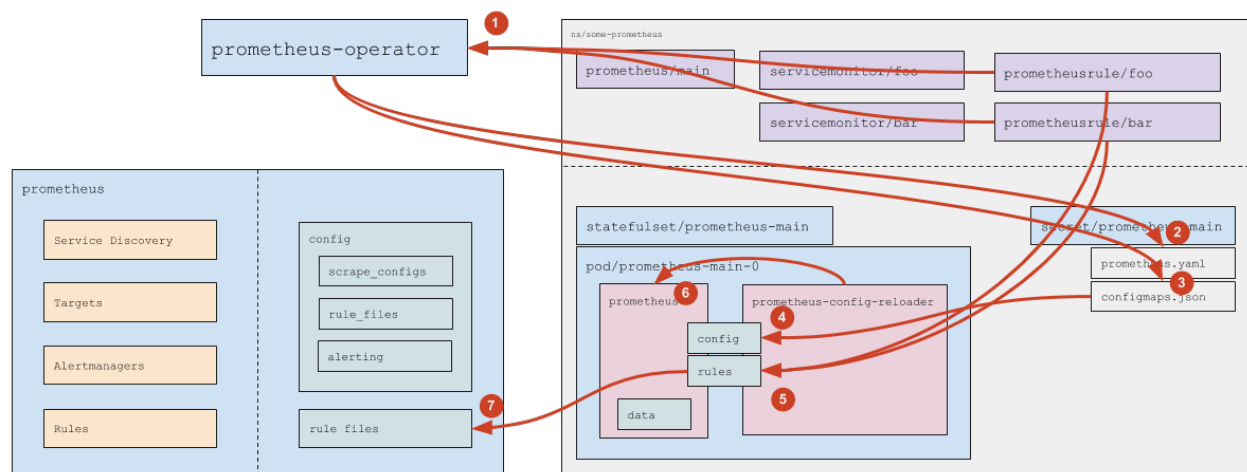
- Три контейнера:
 - `prometheus` — сам Prometheus;
 - `prometheus-config-reloader` — утилита, которая:
 - следит за изменениями `prometheus.yaml` и при необходимости инициирует перезагрузку конфигурации Prometheus (специальным HTTP-запросом, см. подробнее ниже);
 - отслеживает PrometheusRule (см. подробнее ниже), по необходимости скачивает их и перезапускает Prometheus.
 - `kube-rbac-proxy` — прокси для аутентификации и авторизации по RBAC при доступе к метрикам Prometheus.
- Ключевые тома (volumes):
 - `config` — примонтированный секрет с файлами `prometheus.yaml` и `configmaps.json`. Используется в контейнерах `prometheus` и `prometheus-config-reloader`.
 - `rules` — `emptyDir`, который заполняет `prometheus-config-reloader`, а читает `prometheus`. Примонтирован в оба контейнера, в `prometheus` — в режиме только для чтения.
 - `data` — данные Prometheus. Примонтирован только в контейнер `prometheus`.

Обработка ServiceMonitor



1. Prometheus Operator отслеживает ресурсы ServiceMonitor (какие именно — задаётся в спецификации ресурса Prometheus, см. подробнее в официальной документации).
2. Для каждого ServiceMonitor, у которого не задан явный список пространств имён (используется `any: true`), Prometheus Operator через API Kubernetes вычисляет список пространств имён с сервисами (Service), подходящими под лейблы из ServiceMonitor.
3. На основе найденных ресурсов ServiceMonitor и вычисленных пространств имён Prometheus Operator формирует фрагмент конфигурации Prometheus (секцию `scrape_configs`) и сохраняет его в соответствующий секрет.
4. Механизм монтирования Kubernetes обновляет файлы в поде Prometheus, в результате чего обновляется `prometheus.yaml` из секрета.
5. Контейнер `prometheus-config-reloader` обнаруживает изменение `prometheus.yaml` и отправляет Prometheus HTTP-запрос на перезагрузку конфигурации.
6. Prometheus перечитывает конфигурацию и применяет изменения в `scrape_configs` согласно своей логике работы.

Обработка кастомных ресурсов с правилами



1. Prometheus Operator отслеживает объекты PrometheusRule, подходящие под `ruleSelector`, заданный в ресурсе Prometheus.
2. При появлении нового или удалении существующего PrometheusRule Prometheus Operator обновляет `prometheus.yaml` — далее срабатывает та же цепочка, что описана для обработки ServiceMonitor.
3. При добавлении, удалении или изменении содержимого PrometheusRule Prometheus Operator обновляет ConfigMap `prometheus-main-rulefiles-0`.
4. Механизм монтирования Kubernetes доставляет обновлённый ConfigMap в под.
5. Контейнер `prometheus-config-reloader` фиксирует изменения, затем:
 - выгружает обновлённые ConfigMap в каталог `rules` (`emptyDir`);
 - отправляет Prometheus HTTP-запрос на перезагрузку конфигурации.
6. Prometheus перечитывает конфигурацию и применяет изменённые правила.

Настройки

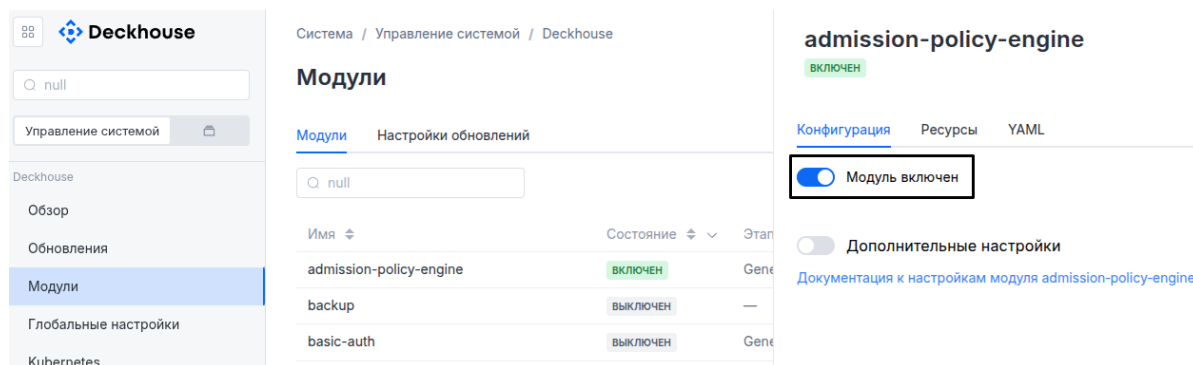
В модуле реализовано 4 алерта.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `operator-prometheus`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `operator-prometheus`:

```
d8 system module enable operator-prometheus
```

- **С помощью ModuleConfig operator-prometheus.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig operator-prometheus` (создайте его, при необходимости).

Пример манифеста для включения модуля `operator-prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-prometheus
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `operator-prometheus`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация

Ресурсы

YAML



Модуль включен



Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация

YAML

Вы также можете отредактировать объект ModuleConfig `operator-prometheus` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `operator-prometheus`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `operator-prometheus` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc operator-prometheus
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `operator-prometheus`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `operator-prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-prometheus
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Custom resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

PodMonitor

Short names: pmon

Scope: Namespaced

Version: v1

PodMonitor определяет мониторинг для набора подов.

- **apiVersion**

строка

APIVersion определяет версию этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

- **kind**

строка

Kind — это строковое значение, представляющее ресурс REST, который представляет этот объект. Серверы могут определять это из эндпоинта, к которому клиент отправляет запросы. Не может быть обновлено. Используется CamelCase. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metadata**

объект

- **spec**

объект

Обязательный параметр

Спецификация желаемого выбора подов для обнаружения целей Prometheus.

- **spec.attachMetadata**

объект

Присоединяет метаданные узла к обнаруженным целям. Требуется Prometheus версии не ниже v2.35.0.

- **spec.attachMetadata.node**

булевый

Если установлено в true, Prometheus должен иметь разрешения на получение узлов.

- **spec.jobLabel**

строка

Лейбл, используемый для получения имени задачи.

- **spec.keepDroppedTargets**

целочисленный

Предел на количество целей, отклоненных переименованием, которые будут оставаться в памяти. 0 означает отсутствие ограничения. Требуется Prometheus >= v2.47.0.

- **spec.labelLimit**

целочисленный

Предел на количество лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelNameLengthLimit**

целочисленный

Предел на длину имён лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelValueLengthLimit**

целочисленный

Предел на длину значений лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.namespaceSelector**

объект

Селектор для выбора, из каких пространств имён обнаруживать объекты Endpoints.

- **spec.namespaceSelector.any**

булевый

Логическое значение, описывающее, выбраны ли все пространства имён в отличие от списка, ограничивающего их.

- **spec.namespaceSelector.matchNames**

массив строк

Список имён пространств имён для выбора.

- **spec.podMetricsEndpoints**

массив объектов

Обязательный параметр

Список эндпоинтов, разрешённых в рамках этого PodMonitor.

PodMetricsEndpoint определяет эндпоинт Kubernetes Pod, поддерживающий сканирование метрик Prometheus.

- **spec.podMetricsEndpoints.authorization**

объект

Раздел авторизации для этого эндпоинта.

- **spec.podMetricsEndpoints.authorization.credentials**
объект
Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.
 - **spec.podMetricsEndpoints.authorization.credentials.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.authorization.credentials.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.authorization.credentials.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.podMetricsEndpoints.authorization.type**
строка
Определяет тип аутентификации. Значение нечувствительно к регистру. "Basic" не поддерживается. По умолчанию: "Bearer"
- **spec.podMetricsEndpoints.basicAuth**
объект
BasicAuth позволяет эндпоинту аутентифицироваться с использованием базовой аутентификации. Подробнее:
<https://prometheus.io/docs/operating/configuration/#endpoint>
 - **spec.podMetricsEndpoints.basicAuth.password**
объект
Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.
 - **spec.podMetricsEndpoints.basicAuth.password.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.basicAuth.password.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.basicAuth.password.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
 - **spec.podMetricsEndpoints.basicAuth.username**
объект
Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.
 - **spec.podMetricsEndpoints.basicAuth.username.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.basicAuth.username.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.basicAuth.username.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.podMetricsEndpoints.bearerTokenSecret**
объект
Секрет для монтирования, чтобы считывать маркер доступа для целей сканирования. Секрет должен находиться в том же пространстве имён, что и мониторинг подов, и быть доступным для оператора Prometheus.
 - **spec.podMetricsEndpoints.bearerTokenSecret.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.bearerTokenSecret.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.bearerTokenSecret.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.podMetricsEndpoints.enableHttp2**
булевый
Включить ли HTTP2.

- **spec.podMetricsEndpoints.filterRunning**
булевый

Исключать поды, которые не работают. (Неудавшиеся, Завершенные). Включено по умолчанию. Подробнее: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-phase>

- **spec.podMetricsEndpoints.followRedirects**
булевый

FollowRedirects настраивает, следуют ли запросы на сканирование перенаправлениям HTTP 3xx.

- **spec.podMetricsEndpoints.honorLabels**
булевый

HonorLabels выбирает лейблы метрик при столкновении с целевыми лейблами.

- **spec.podMetricsEndpoints.honorTimestamps**
булевый

HonorTimestamps управляет тем, соблюдает ли Prometheus временные лейблы, присутствующие в собранных данных.

- **spec.podMetricsEndpoints.interval**
строка

Интервал, на котором должны собираться метрики. Если не указано, используется глобальный интервал сбора Prometheus.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

- **spec.podMetricsEndpoints.metricRelabelings**
массив объектов

MetricRelabelConfigs для применения к выборкам до ввода данных.

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.podMetricsEndpoints.metricRelabelings.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: "Replace"

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.podMetricsEndpoints.metricRelabelings.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.podMetricsEndpoints.metricRelabelings.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.podMetricsEndpoints.metricRelabelings.replacement**
строка

Значение замены для действия Replace, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.podMetricsEndpoints.metricRelabelings.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (SourceLabels).

- **spec.podMetricsEndpoints.metricRelabelings.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

LabelName — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.podMetricsEndpoints.metricRelabelings.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.podMetricsEndpoints.oauth2**
объект

OAuth2 для URL. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.podMetricsEndpoints.oauth2.clientId**
объект

Обязательный параметр

Секрет или ConfigMap, содержащие идентификатор клиента OAuth2.

- **spec.podMetricsEndpoints.oauth2.clientId.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.podMetricsEndpoints.oauth2.clientId.configMap.key**
строка
Обязательный параметр
Ключ для выбора.
- **spec.podMetricsEndpoints.oauth2.clientId.configMap.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
- **spec.podMetricsEndpoints.oauth2.clientId.configMap.optional**
булевый
Укажите, должен ли быть определён ConfigMap или его ключ.
- **spec.podMetricsEndpoints.oauth2.clientId.secret**
объект
Секрет, содержащий данные для использования в целях.
 - **spec.podMetricsEndpoints.oauth2.clientId.secret.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.oauth2.clientId.secret.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.oauth2.clientId.secret.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.podMetricsEndpoints.oauth2.clientSecret**
объект
Обязательный параметр
Секрет, содержащий секрет клиента OAuth2.
 - **spec.podMetricsEndpoints.oauth2.clientSecret.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.podMetricsEndpoints.oauth2.clientSecret.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.podMetricsEndpoints.oauth2.clientSecret.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.podMetricsEndpoints.oauth2.endpointParams**
объект
Параметры для добавления к URL токена.
- **spec.podMetricsEndpoints.oauth2.scopes**
массив строк
Области OAuth2, используемые для запроса токена.
- **spec.podMetricsEndpoints.oauth2.tokenUrl**
строка
Обязательный параметр
URL для получения токена.
- **spec.podMetricsEndpoints.params**
объект
Необязательные параметры URL HTTP.
- **spec.podMetricsEndpoints.path**
строка
HTTP-путь для сбора метрик. Если не указано, Prometheus использует значение по умолчанию (например, `/metrics`).
- **spec.podMetricsEndpoints.port**
строка
Имя порта пода, к которому относится этот эндпоинт. Взаимоисключается с targetPort.
- **spec.podMetricsEndpoints.proxyUrl**
строка
ProxyURL, например `http://proxyserver:2195`. Направляет запросы на сканирование через прокси посредством этого эндпоинта.
- **spec.podMetricsEndpoints.relabelings**
массив объектов
RelabelConfigs для применения к выборкам перед сбором. Оператор Prometheus автоматически добавляет переименования для некоторых стандартных полей Kubernetes. Имя изначальной задачи сбора доступно через лейбл

`tmp.prometheus.job.name`. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.podMetricsEndpoints.relabelings.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.podMetricsEndpoints.relabelings.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.podMetricsEndpoints.relabelings.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.podMetricsEndpoints.relabelings.replacement**
строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.podMetricsEndpoints.relabelings.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.podMetricsEndpoints.relabelings.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.podMetricsEndpoints.relabelings.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.podMetricsEndpoints.scheme**
строка

HTTP-схема для использования при сканировании. Ожидаемые значения — `http` и `https`, если вы не переписываете лейбл `__scheme__` посредством переименования. Если не указано, Prometheus использует значение по умолчанию — `http`.

Допустимые значения: `http`, `https`

- **spec.podMetricsEndpoints.scrapeTimeout**
строка

Таймаут, после которого сбор данных прекращается. Если не указано, используется глобальный интервал сбора Prometheus.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)??$`

- **spec.podMetricsEndpoints.targetPort**
строка или число

Устарело: используйте `port` вместо этого.

- **spec.podMetricsEndpoints.tlsConfig**
объект

Конфигурация TLS для использования при сканировании с эндпоинта.

- **spec.podMetricsEndpoints.tlsConfig.ca**
объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.podMetricsEndpoints.tlsConfig.ca.configMap**
объект

`ConfigMap`, содержащий данные для использования в целях.

- **spec.podMetricsEndpoints.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.podMetricsEndpoints.tlsConfig.ca.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.podMetricsEndpoints.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.podMetricsEndpoints.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.podMetricsEndpoints.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.podMetricsEndpoints.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.podMetricsEndpoints.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.podMetricsEndpoints.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.podMetricsEndpoints.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.podMetricsEndpoints.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.podMetricsEndpoints.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.podMetricsEndpoints.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.podMetricsEndpoints.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.podMetricsEndpoints.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.podMetricsEndpoints.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.podMetricsEndpoints.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.podMetricsEndpoints.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.podMetricsEndpoints.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.podMetricsEndpoints.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.podMetricsEndpoints.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.podMetricsEndpoints.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.podMetricsEndpoints.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей.
- **spec.podTargetLabels**
массив строк

PodTargetLabels переносят лейблы с Kubernetes Pod на цель.
- **spec.sampleLimit**
целочисленный

SampleLimit определяет предел на количество выборок, которое будет принято за одно сканирование.
- **spec.selector**
объект

Обязательный параметр

Селектор для выбора объектов типа Pod.
 - **spec.selector.matchExpressions**
массив объектов

matchExpressions — это список требований селектора лейблов. Требования объединены с помощью логического оператора И (AND).

Требование селектора лейблов — это селектор, который содержит значения, ключ и оператор, связывающий ключ и значения.
 - **spec.selector.matchExpressions.key**
строка

Обязательный параметр

key — это ключ лейбла, к которому применяется селектор.
 - **spec.selector.matchExpressions.operator**
строка

Обязательный параметр

operator представляет взаимоотношение ключа с набором значений. Допустимые операторы: In, NotIn, Exists и DoesNotExist.
 - **spec.selector.matchExpressions.values**
массив строк

values — это массив строковых значений. Если оператор In или NotIn, массив values должен быть непустым. Если оператор Exists или DoesNotExist, массив values должен быть пустым. Этот массив заменяется во время стратегического объединения патчей.
 - **spec.selector.matchLabels**
объект

matchLabels — это карта пар {ключ, значение}. Один {ключ, значение} в карте matchLabels эквивалентен элементу matchExpressions, у которого поле ключа — “key”, оператор — “In”, а массив values содержит только “value”. Требования объединены с помощью логического оператора И (AND).
- **spec.targetLimit**
целочисленный

TargetLimit определяет предел количества целей, которые будут приняты для сбора.

Probe

Short names: prb

Scope: Namespaced
Version: v1

Probe определяет мониторинг для набора статических целей или ingress.

- **apiVersion**
строка

APIVersion определяет версию этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>
- **kind**
строка

Kind — это строковое значение, представляющее ресурс REST, который представляет этот объект. Серверы могут определять это из эндпоинта, к которому клиент отправляет запросы. Не может быть обновлено. Используется CamelCase. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>
- **metadata**
объект
- **spec**
объект

Обязательный параметр

Спецификация желаемого выбора ingress для обнаружения целей Prometheus.
 - **spec.authorization**
объект

Раздел авторизации для этого эндпоинта.
 - **spec.authorization.credentials**
объект

Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.
 - **spec.authorization.credentials.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.authorization.credentials.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.authorization.credentials.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.authorization.type**

строка

Определяет тип аутентификации. Значение нечувствительно к регистру. "Basic" не поддерживается. По умолчанию: "Bearer"

○ **spec.basicAuth**

объект

BasicAuth позволяет эндпоинту аутентифицироваться с использованием базовой аутентификации. Более подробно: <https://prometheus.io/docs/operating/configuration/#endpoint>

■ **spec.basicAuth.password**

объект

Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.

■ **spec.basicAuth.password.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.basicAuth.password.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.basicAuth.password.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.basicAuth.username**

объект

Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.

■ **spec.basicAuth.username.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.basicAuth.username.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.basicAuth.username.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

○ **spec.bearerTokenSecret**

объект

Секрет для монтирования, чтобы считать токен владельца для целей сканирования. Секрет должен находиться в том же пространстве имён, что и probe, и быть доступным для Prometheus Operator.

■ **spec.bearerTokenSecret.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.bearerTokenSecret.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.bearerTokenSecret.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

○ **spec.interval**

строка

Интервал, на котором цели сканируются с использованием настроенного prob. Если не указано, используется глобальный интервал сбора Prometheus.

Шаблон: `^[0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

○ **spec.jobName**

строка

Имя задания, присваиваемое собранным метрикам по умолчанию.

- **spec.keepDroppedTargets**
целочисленный

Предел на количество целей, отклоненных переименованием, которые будут оставаться в памяти. 0 означает отсутствие ограничения. Требуется Prometheus $\geq$ v2.47.0.

- **spec.labelLimit**
целочисленный

Предел на количество лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelNameLengthLimit**
целочисленный

Предел на длину имени лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelValueLengthLimit**
целочисленный

Предел на длину значений лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.metricRelabelings**
массив объектов

MetricRelabelConfigs для применения к выборкам до ввода данных.

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.metricRelabelings.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.metricRelabelings.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.metricRelabelings.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.metricRelabelings.replacement**
строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.metricRelabelings.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.metricRelabelings.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.metricRelabelings.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.module**
строка

Модуль для использования во время пробинга, указывающий, как осуществлять пробинг цели. Пример настройки модуля в экспортере Blackbox: https://github.com/prometheus/blackbox_exporter/blob/master/example.yml

- **spec.oauth2**
объект

OAuth2 для URL. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.oauth2.clientId**
объект

Обязательный параметр

Секрет или ConfigMap, содержащие клиентский идентификатор OAuth2.

- **spec.oauth2.clientId.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.oauth2.clientId.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.oauth2.clientId.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.oauth2.clientId.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.oauth2.clientId.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.oauth2.clientId.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.oauth2.clientId.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.oauth2.clientId.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.oauth2.clientSecret**
объект

Обязательный параметр

Секрет, содержащий клиентский секрет OAuth2.

- **spec.oauth2.clientSecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.oauth2.clientSecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.oauth2.clientSecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.oauth2.endpointParams**
объект

Параметры для добавления к URL токена.

- **spec.oauth2.scopes**
массив строк

Области OAuth2, используемые для запроса токена.

- **spec.oauth2.tokenUrl**
строка

Обязательный параметр

URL для получения токена.

- **spec.prober**
объект

Спецификация для prober для использования пробиры целей. Параметр prober.URL обязателен. Цели не могут быть пробируются, если не указаны.

- **spec.prober.path**
строка

Путь для сбора метрик. По умолчанию `/probe`.

По умолчанию: `/probe`

- **spec.prober.proxyUrl**
строка

Необязательный ProxyURL.

- **spec.prober.scheme**
строка

HTTP-схема для использования при сборе данных. Ожидаемые значения — `http` и `https`, если вы не переписываете лейбл `__scheme__` посредством переименования. Если не указано, Prometheus использует значение по умолчанию — `http`.

Допустимые значения: `http`, `https`

- **spec.prober.url**
строка

Обязательный параметр

Обязательный URL для prob.

- **spec.sampleLimit**
целочисленный

Предел на количество выборок, которое будет принято за одно сканирование.

- **spec.scrapeTimeout**
строка

Таймаут для сбора метрик с экспортера Prometheus. Если не указано, используется глобальный таймаут сбора Prometheus.

Шаблон: `^([0](([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?)?$`

- **spec.targetLimit**
целочисленный

Предел количества целей, которые будут приняты для сбора.

- **spec.targets**
объект

Targets определяет набор статических или динамически обнаруживаемых целей для пробинга.

- **spec.targets.ingress**
объект

ingress определяет объекты Ingress для пробинга и конфигурацию переименования. Если также определен `staticConfig`, он имеет приоритет.

- **spec.targets.ingress.namespaceSelector**
объект

Из каких пространств имён выбирать объекты Ingress.

- **spec.targets.ingress.namespaceSelector.any**
булевый

Логическое значение, описывающее, выбраны ли все пространства имён в отличие от списка, ограничивающего их.

- **spec.targets.ingress.namespaceSelector.matchNames**
массив строк

Список имён пространств имён для выбора.

- **spec.targets.ingress.relabelingConfigs**
массив объектов

RelabelConfigs для применения к набору лейблов цели до их сбора. Изначальный адрес ingress доступен через лейбл `__tmp_prometheus_ingress_address`. Он может быть использован для настройки пробируемого URL. Имя изначальной задачи сбора доступно через лейбл `__tmp_prometheus_job_name`. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.targets.ingress.relabelingConfigs.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.targets.ingress.relabelingConfigs.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.targets.ingress.relabelingConfigs.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.targets.ingress.relabelingConfigs.replacement**
строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.targets.ingress.relabelingConfigs.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.targets.ingress.relabelingConfigs.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

LabelName — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.targets.ingress.relabelingConfigs.targetLabel**
строка

Лейбл, в которую записывается результирующая строка при замене. Она обязательна для действий Replace, HashMod, Lowercase, Uppercase, KeepEqual и DropEqual. Доступны группы захвата из регулярного выражения.

- **spec.targets.ingress.selector**
объект

Селектор для выбора объектов Ingress.

- **spec.targets.ingress.selector.matchExpressions**
массив объектов

matchExpressions — это список требований для селектора лейблов. Эти требования объединяются с помощью логического оператора И (AND).

Требование селектора лейблов — это селектор, который содержит значения, ключ и оператор, связывающий ключ и значения.

- **spec.targets.ingress.selector.matchExpressions.key**
строка

Обязательный параметр

key — это ключ лейбла, к которому применяется селектор.

- **spec.targets.ingress.selector.matchExpressions.operator**
строка

Обязательный параметр

operator представляет взаимоотношение ключа с набором значений. Допустимые операторы: In, NotIn, Exists и DoesNotExist.

- **spec.targets.ingress.selector.matchExpressions.values**
массив строк

values — это массив строковых значений. Если оператор In или NotIn, массив values должен быть непустым. Если оператор Exists или DoesNotExist, массив values должен быть пустым. Этот массив заменяется во время стратегического объединения патчей.

- **spec.targets.ingress.selector.matchLabels**
объект

matchLabels — это отображение пар {ключ, значение}. Одиночная пара {ключ, значение} в matchLabels эквивалентна элементу matchExpressions, у которого поле ключа — “key”, оператор — “In”, а массив значений содержит только “value”. Эти требования объединяются с помощью логического оператора И (AND).

- **spec.targets.staticConfig**
объект

staticConfig определяет статический список целей для пробинга и конфигурации переименования. Если также определен ingress, staticConfig имеет приоритет. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#static_config.

- **spec.targets.staticConfig.labels**
объект

Лейблы, присваиваемые всем метрикам, полученным с целей.

- **spec.targets.staticConfig.relabelingConfigs**
массив объектов

RelabelConfigs для применения к набору лейблов целей до их сбора. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.targets.staticConfig.relabelingConfigs.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия Uppercase и Lowercase требуют Prometheus версии не ниже v2.36.0. Действия DropEqual и KeepEqual требуют Prometheus версии не ниже v2.41.0. По умолчанию: “Replace”

По умолчанию: replace

Допустимые значения: replace, Replace, keep, Keep, drop, Drop, hashmod, HashMod, labelmap, LabelMap, labeledrop, LabelDrop, labelkeep, LabelKeep, lowercase, Lowercase, uppercase, Uppercase, keepequal, KeepEqual, dropequal, DropEqual

- **spec.targets.staticConfig.relabelingConfigs.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — HashMod.

- **spec.targets.staticConfig.relabelingConfigs.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.targets.staticConfig.relabelingConfigs.replacement**
строка

Значение замены для действия Replace, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.targets.staticConfig.relabelingConfigs.separator**

строка

Разделитель — это строка между объединенными исходными лейблами (SourceLabels).

- **spec.targets.staticConfig.relabelingConfigs.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

LabelName — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.targets.staticConfig.relabelingConfigs.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.targets.staticConfig.static**
массив строк

Список хостов для пробинга.

◦ **spec.tlsConfig** объект

Конфигурация TLS для использования при сборе данных с эндпоинта.

- **spec.tlsConfig.ca**
объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.tlsConfig.ca.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей.

ScrapeConfig

Short names: scfg

Scope: Namespaced
Version: v1alpha1

ScrapeConfig определяет пространственный Prometheus scrape_config, который агрегируется через несколько пространств имён в конфигурации Prometheus.

- **apiVersion**
строка

APIVersion определяет версию этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

- **kind**
строка

Kind — это строковое значение, представляющее ресурс REST, который представляет этот объект. Серверы могут определять это из эндпоинта, к которому клиент отправляет запросы. Не может быть обновлено. Используется CamelCase. Более подробно: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metadata**
объект

- **spec**
объект

Обязательный параметр

ScrapeConfigSpec — это спецификация желаемой конфигурации для конфигурации сбора.

- **spec.authorization**
объект

Заголовок авторизации для использования в каждом запросе сбора.

- **spec.authorization.credentials**
объект

Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.

- **spec.authorization.credentials.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.authorization.credentials.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.authorization.credentials.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.authorization.type**
строка

Определяет тип аутентификации. Значение нечувствительно к регистру. “Basic” не поддерживается. По умолчанию: “Bearer”

- **spec.basicAuth**
объект

Информация BasicAuth для использования в каждом запросе сбора.

- **spec.basicAuth.password**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.

- **spec.basicAuth.password.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.basicAuth.password.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.basicAuth.password.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.basicAuth.username**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.

- **spec.basicAuth.username.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.basicAuth.username.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.basicAuth.username.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs**
массив объектов

ConsulSDConfigs определяет список конфигураций обнаружения сервисов Consul.

ConsulSDConfig определяет конфигурацию обнаружения сервисов Consul. Смотри подробнее:
https://prometheus.io/docs/prometheus/latest/configuration/configuration/#consul_sd_config

- **spec.consulSDConfigs.allow_stale**
булевый

Разрешить устаревшие результаты Consul (смотри <https://www.consul.io/api/features/consistency.html>). Снизит нагрузку на Consul. Если не указано, Prometheus использует свое значение по умолчанию.

- **spec.consulSDConfigs.authorization**
объект

Конфигурация заголовка авторизации для аутентификации на сервере Consul.

- **spec.consulSDConfigs.authorization.credentials**
объект

Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.

- **spec.consulSDConfigs.authorization.credentials.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.authorization.credentials.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.authorization.credentials.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.authorization.type**
строка

Определяет тип аутентификации. Значение нечувствительно к регистру. "Basic" не поддерживается. По умолчанию: "Bearer"

- **spec.consulSDConfigs.basicAuth**
объект

Информация BasicAuth для аутентификации на сервере Consul. Подробнее:
<https://prometheus.io/docs/operating/configuration/#endpoints>

- **spec.consulSDConfigs.basicAuth.password**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.

- **spec.consulSDConfigs.basicAuth.password.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.basicAuth.password.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.basicAuth.password.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.basicAuth.username**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.

- **spec.consulSDConfigs.basicAuth.username.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.basicAuth.username.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.basicAuth.username.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.datacenter**
строка

Имя центра обработки данных Consul, если не указано, будет использован локальный агент Consul.

- **spec.consulSDConfigs.enable_http2**
булевый

Включить ли HTTP2. Если не указано, Prometheus использует свое значение по умолчанию.

- **spec.consulSDConfigs.follow_redirects**
булевый

Настраивает следование HTTP-запросов за перенаправлениями HTTP 3xx. Если не указано, Prometheus использует свое значение по умолчанию.

- **spec.consulSDConfigs.namespace**
строка

Пространства имён поддерживаются только в Consul Enterprise.

- **spec.consulSDConfigs.no_proxy**
строка

Строка с разделением запятыми, содержащая IP, нотации CIDR, доменные имена, которые должны быть исключены из проксирования. IP и доменные имена могут содержать номера портов.

- **spec.consulSDConfigs.node_meta**
объект

Ключ/значение метаданных узла для фильтрации узлов для данного сервиса.

- **spec.consulSDConfigs.oauth2**
объект

Дополнительная конфигурация OAuth 2.0.

- **spec.consulSDConfigs.oauth2.clientId**
объект

Обязательный параметр

Секрет или ConfigMap, содержащие идентификатор клиента OAuth2

- **spec.consulSDConfigs.oauth2.clientId.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.consulSDConfigs.oauth2.clientId.configMap.key**
строка
Обязательный параметр
Ключ для выбора.
- **spec.consulSDConfigs.oauth2.clientId.configMap.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
- **spec.consulSDConfigs.oauth2.clientId.configMap.optional**
булевый
Укажите, должен ли быть определён ConfigMap или его ключ.
- **spec.consulSDConfigs.oauth2.clientId.secret**
объект
Секрет, содержащий данные для использования в целях.
 - **spec.consulSDConfigs.oauth2.clientId.secret.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.consulSDConfigs.oauth2.clientId.secret.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.consulSDConfigs.oauth2.clientId.secret.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.consulSDConfigs.oauth2.clientSecret**
объект
Обязательный параметр
Секрет, содержащий секрет клиента OAuth2
 - **spec.consulSDConfigs.oauth2.clientSecret.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.consulSDConfigs.oauth2.clientSecret.name**
строка
Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.consulSDConfigs.oauth2.clientSecret.optional**
булевый
Укажите, должен ли быть определён Secret или его ключ.
- **spec.consulSDConfigs.oauth2.endpointParams**
объект
Параметры для добавления к URL токена
- **spec.consulSDConfigs.oauth2.scopes**
массив строк
Области OAuth2, используемые для запроса токена
- **spec.consulSDConfigs.oauth2.tokenUrl**
строка
Обязательный параметр
URL для получения токена
- **spec.consulSDConfigs.partition**
строка
Административные разделы поддерживаются только в Consul Enterprise.
- **spec.consulSDConfigs.proxy_connect_header**
объект
Указывает заголовки для отправки прокси-серверам во время CONNECT-запросов.
 - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>**
объект
<KEY_NAME> — имя параметра (ключ объекта).
SecretKeySelector выбирает ключ из Секрета.
 - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.key**
строка
Обязательный параметр
Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.proxy_from_environment**
булевый

Использовать URL прокси, указанный переменными окружения (HTTP_PROXY, https_proxy, HTTPS_PROXY, https_proxy и по_проху). Если не указано, Prometheus использует свое значение по умолчанию.

- **spec.consulSDConfigs.proxy_url**
строка

Необязательный URL прокси.

- **spec.consulSDConfigs.refresh_interval**
строка

Время после которого предоставленные имена обновляются. В больших установках имеет смысл увеличить это значение, так как каталог постоянно обновляется. Если не указано, Prometheus использует свое значение по умолчанию.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?`

- **spec.consulSDConfigs.scheme**
строка

По умолчанию схема HTTP "http"

Допустимые значения: HTTP, HTTPS

- **spec.consulSDConfigs.server**
строка

Обязательный параметр

Допустимая строка, состоящая из имени хоста или IP, за которым следует необязательный номер порта.

- **spec.consulSDConfigs.services**
массив строк

Список сервисов, для которых извлекаются цели. Если пропущено, извлекаются все сервисы.

- **spec.consulSDConfigs.tag_separator**
строка

Строка, которой соединяются теги Consul в лейбле тегов. Если не указано, Prometheus использует свое значение по умолчанию.

- **spec.consulSDConfigs.tags**
массив строк

Необязательный список тегов, используемых для фильтрации узлов для данного сервиса. Услуги должны содержать все теги в списке.

- **spec.consulSDConfigs.tlsConfig**
объект

Конфигурация TLS

- **spec.consulSDConfigs.tlsConfig.ca**
объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.consulSDConfigs.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.consulSDConfigs.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.consulSDConfigs.tlsConfig.ca.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.consulSDConfigs.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.consulSDConfigs.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.consulSDConfigs.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.consulSDConfigs.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.consulSDConfigs.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.consulSDConfigs.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.consulSDConfigs.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.consulSDConfigs.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.consulSDConfigs.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.consulSDConfigs.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей.

- **spec.consulSDConfigs.tokenRef**
объект

Ссылка на токен ACL Consul, если не предоставлено, будет использоваться ACL от локального агента Consul.

- **spec.consulSDConfigs.tokenRef.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.consulSDConfigs.tokenRef.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.consulSDConfigs.tokenRef.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.dnsSDConfigs**

массив объектов

DNSSDConfigs определяет список конфигураций обнаружения сервисов DNS.

DNSSDConfig позволяет указывать набор доменных имён DNS, которые периодически запрашиваются для обнаружения списка целей. DNS-серверы для контакта читаются из /etc/resolv.conf. Смотри подробнее: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#dns_sd_config

- **spec.dnsSDConfigs.names**

массив строк

Обязательный параметр

Список доменных имён DNS для запроса.

- **spec.dnsSDConfigs.port**

целочисленный

Номер порта, используемый, если тип запроса не SRV. Игнорируется для записей SRV.

- **spec.dnsSDConfigs.refreshInterval**

строка

RefreshInterval настраивает время после которого предоставленные имена обновляются. Если не задано, Prometheus использует свое значение по умолчанию.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w?([0-9]+)d?([0-9]+)h?([0-9]+)m?([0-9]+)s?([0-9]+)ms)?$`

- **spec.dnsSDConfigs.type**

строка

Тип запроса DNS, который выполнить. Один из SRV, A, AAAA или MX. Если не задано, Prometheus использует свое значение по умолчанию.

Допустимые значения: `SRV`, `A`, `AAAA`, `MX`

- **spec.fileSDConfigs**

массив объектов

FileSDConfigs определяет список конфигураций обнаружения сервисов в файле.

FileSDConfig определяет конфигурацию обнаружения сервисов в файле Prometheus. Смотри подробнее: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#file_sd_config

- **spec.fileSDConfigs.files**

массив строк

Обязательный параметр

Список файлов для использования в обнаружении по файлам. Рекомендация: использовать абсолютные пути. Хотя относительные пути работают, проект prometheus-operator не дает никаких гарантий относительно рабочего каталога, где хранится конфигурационный файл. Файлы должны быть смонтированы с использованием Prometheus.ConfigMaps или Prometheus.Secrets.

- **Элемент массива**

строка

SDFile представляет собой файл, используемый для обнаружения сервиса

Шаблон: `^[^*]*(\[*/\])?\.(json|yaml|yml|JSON|YML|YAML)$`

- **spec.fileSDConfigs.refreshInterval**

строка

RefreshInterval настраивает интервал обновления, в котором Prometheus будет перезагружать содержимое файлов.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w?([0-9]+)d?([0-9]+)h?([0-9]+)m?([0-9]+)s?([0-9]+)ms)?$`

- **spec.honorLabels**

булевый

HonorLabels выбирает лейблы метрик при столкновении с целевыми лейблами.

- **spec.honorTimestamps**

булевый

HonorTimestamps управляет тем, соблюдать ли Prometheus временные лейблы, присутствующие в собранных данных.

- **spec.httpSDConfigs**

массив объектов

HTTPSDConfigs определяет список конфигураций обнаружения сервисов HTTP.

HTTPSDConfig определяет конфигурацию обнаружения сервисов HTTP Prometheus. Смотри подробнее: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#http_sd_config

- **spec.httpSDConfigs.authorization**

объект

Конфигурация заголовка авторизации для аутентификации на целевом эндпоинте HTTP.

- **spec.httpSDConfigs.authorization.credentials**

объект

Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.

- **spec.httpSDConfigs.authorization.credentials.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.authorization.credentials.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.authorization.credentials.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.authorization.type**
строка

Определяет тип аутентификации. Значение нечувствительно к регистру. “Basic” не поддерживается. По умолчанию: “Bearer”

- **spec.httpSDConfigs.basicAuth**
объект

Информация BasicAuth для аутентификации на целевом эндпоинте HTTP. Подробнее:
<https://prometheus.io/docs/operating/configuration/#endpoints>

- **spec.httpSDConfigs.basicAuth.password**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.

- **spec.httpSDConfigs.basicAuth.password.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.basicAuth.password.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.basicAuth.password.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.basicAuth.username**
объект

Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.

- **spec.httpSDConfigs.basicAuth.username.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.basicAuth.username.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.basicAuth.username.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.refreshInterval**
строка

RefreshInterval настраивает интервал обновления, в котором Prometheus будет запрашивать эндпоинт для обновления списка целей.

Шаблон: `^[0|([0-9]+)y)?([0-9]+w)?([0-9]+d)?([0-9]+h)?([0-9]+m)?([0-9]+s)?([0-9]+ms)?$`

- **spec.httpSDConfigs.tlsConfig**
объект

TLS-конфигурация, применяемая к целевому эндпоинту HTTP.

- **spec.httpSDConfigs.tlsConfig.ca**
объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.httpSDConfigs.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.httpSDConfigs.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.httpSDConfigs.tlsConfig.ca.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.httpSDConfigs.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.httpSDConfigs.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.httpSDConfigs.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.httpSDConfigs.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.httpSDConfigs.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.httpSDConfigs.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.httpSDConfigs.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.httpSDConfigs.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.httpSDConfigs.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.httpSDConfigs.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.httpSDConfigs.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.httpSDConfigs.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей.

- **spec.httpSDConfigs.url**
строка

Обязательный параметр

URL, из которого извлекаются цели.

Шаблон: `^http(s)?://.+`

- **spec.keepDroppedTargets**
целочисленный

Предел на количество целей, отклоненных переименованием, которые будут оставаться в памяти. 0 означает отсутствие ограничения. Требуется Prometheus $\geq$ v2.47.0.

- **spec.kubernetesSDConfigs**
массив объектов

KubernetesSDConfigs определяет список конфигураций обнаружения сервисов Kubernetes.

KubernetesSDConfig позволяет получать цели для сбора из REST API Kubernetes. См. подробнее: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

- **spec.kubernetesSDConfigs.role**
строка

Обязательный параметр

Роль сущностей Kubernetes, которые должны быть обнаружены. В настоящее время поддерживается только роль "Node".

Допустимые значения: `Node`

- **spec.labelLimit**
целочисленный

Предел на количество лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelNameLengthLimit**
целочисленный

Предел на длину имени лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelValueLengthLimit**
целочисленный

Предел на длину значений лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.metricRelabelings**
массив объектов

MetricRelabelConfigs для применения к выборкам до ввода данных.

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.metricRelabelings.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: "Replace"

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.metricRelabelings.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.metricRelabelings.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.metricRelabelings.replacement**
строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.metricRelabelings.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.metricRelabelings.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*`

- **spec.metricRelabelings.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.metricsPath**
строка

HTTP-путь для сбора метрик. Если не указано, Prometheus использует значение по умолчанию (например, /metrics).

- **spec.params**
объект

Необязательные параметры URL HTTP

- **spec.relabelings**
массив объектов

RelabelConfigs определяют, как переписывать лейблы целей перед их сбором. Оператор Prometheus автоматически добавляет переименования для некоторых стандартных полей Kubernetes. Имя изначальной задачи сбора доступно через лейбл `__tmp_prometheus_job_name`. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Более подробно: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.relabelings.action**
строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.relabelings.modulus**
целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.relabelings.regex**
строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.relabelings.replacement**
строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.relabelings.separator**
строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.relabelings.sourceLabels**
массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**
строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.relabelings.targetLabel**
строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.sampleLimit**
целочисленный

`SampleLimit` определяет предел на количество выборок, которое будет принято за одно сканирование.

- **spec.scheme**
строка

Настраивает схему протокола, используемую для запросов. Если не указано, Prometheus по умолчанию использует HTTP.

Допустимые значения: `HTTP`, `HTTPS`

- **spec.scrapeInterval**
строка

`ScrapeInterval` — это интервал между последовательными сборками.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

- **spec.scrapeTimeout**
строка

`ScrapeTimeout` — это количество секунд, в течение которого запрос на сбор данных считается истекшим.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

- **spec.staticConfigs**
массив объектов

`StaticConfigs` определяет список статических целей с общим набором лейблов.

`StaticConfig` определяет статическую конфигурацию Prometheus. Смотри подробнее: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#scrape_config

- **spec.staticConfigs.labels**
объект

Лейблы, присваиваемые всем метрикам, полученным с целей.

- **spec.staticConfigs.targets**
массив строк

Список целей для этой статической конфигурации.

- **Элемент массива**
строка

Target представляет собой цель для сбора Prometheus

- **spec.targetLimit**
целочисленный

TargetLimit определяет предел количества целей, которые будут приняты для сбора.

- **spec.tlsConfig**
объект

Конфигурация TLS для использования в каждом запросе сбора

- **spec.tlsConfig.ca**
объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.tlsConfig.ca.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей

ServiceMonitor

Short names: smon

Scope: Namespaced

Version: v1

ServiceMonitor определяет мониторинг для набора сервисов.

- **apiVersion**
строка

APIVersion определяет версию этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения. Более подробно:
<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

- **kind**
строка

Kind — это строковое значение, представляющее ресурс REST, который представляет этот объект. Серверы могут определять это из эндпоинта, к которому клиент отправляет запросы. Не может быть обновлено. Используется CamelCase. Более подробно:
<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metadata**
объект

- **spec**
объект

Обязательный параметр

Спецификация желаемого выбора сервиса для обнаружения целей Prometheus.

- **spec.attachMetadata**
объект

Присоединяет метаданные узла к обнаруженным целям. Требуется Prometheus версии не ниже v2.37.0.

- **spec.attachMetadata.node**
булевый

Если установлено в true, Prometheus должен иметь разрешения на получение узлов.

- **spec.endpoints**
массив объектов

Обязательный параметр

Список эндпоинтов, разрешенных в рамках этого ServiceMonitor.

Определяет эндпоинт, поддерживающий сканирование метрик Prometheus.

- **spec.endpoints.authorization**
объект

Раздел авторизации для этого эндпоинта.

- **spec.endpoints.authorization.credentials**
объект

Выбирает ключ Секрета в пространстве имён, который содержит учетные данные для аутентификации.

- **spec.endpoints.authorization.credentials.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.endpoints.authorization.credentials.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.endpoints.authorization.credentials.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.endpoints.authorization.type**

строка

Определяет тип аутентификации. Значение нечувствительно к регистру. "Basic" не поддерживается. По умолчанию: "Bearer"

■ **spec.endpoints.basicAuth**

объект

BasicAuth позволяет эндпоинту аутентифицироваться с использованием базовой аутентификации. Подробнее: <https://prometheus.io/docs/operating/configuration/#endpoints>

■ **spec.endpoints.basicAuth.password**

объект

Секрет в пространстве имён мониторинга сервиса, который содержит пароль для аутентификации.

■ **spec.endpoints.basicAuth.password.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.endpoints.basicAuth.password.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.endpoints.basicAuth.password.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.endpoints.basicAuth.username**

объект

Секрет в пространстве имён мониторинга сервиса, который содержит имя пользователя для аутентификации.

■ **spec.endpoints.basicAuth.username.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.endpoints.basicAuth.username.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.endpoints.basicAuth.username.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.endpoints.bearerTokenFile**

строка

Файл для считывания маркера доступа (bearer токена) для целей сканирования.

■ **spec.endpoints.bearerTokenSecret**

объект

Секрет для монтирования, чтобы считывать маркер доступа для целей сканирования. Секрет должен находиться в том же пространстве имён, что и мониторинг сервиса, и быть доступным для оператора Prometheus.

■ **spec.endpoints.bearerTokenSecret.key**

строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

■ **spec.endpoints.bearerTokenSecret.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

■ **spec.endpoints.bearerTokenSecret.optional**

булевый

Укажите, должен ли быть определён Secret или его ключ.

■ **spec.endpoints.enableHttp2**

булевый

Включить ли HTTP2.

- **spec.endpoints.filterRunning**

булевый

Исключать поды, которые не работают. (Неудавшиеся, Завершенные). Включено по умолчанию. Подробнее: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-phase>

- **spec.endpoints.followRedirects**

булевый

FollowRedirects настраивает, следуют ли запросы на сканирование перенаправлениям HTTP 3xx.

- **spec.endpoints.honorLabels**

булевый

HonorLabels выбирает лейблы метрик при столкновении с целевыми лейблами.

- **spec.endpoints.honorTimestamps**

булевый

HonorTimestamps управляет тем, соблюдает ли Prometheus временные лейблы, присутствующие в собранных данных.

- **spec.endpoints.interval**

строка

Интервал, на котором должны собираться метрики. Если не указано, используется глобальный интервал сбора Prometheus.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?`

- **spec.endpoints.metricRelabelings**

массив объектов

MetricRelabelConfigs для применения к выборкам до ввода данных.

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.endpoints.metricRelabelings.action**

строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.endpoints.metricRelabelings.modulus**

целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.endpoints.metricRelabelings.regex**

строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.endpoints.metricRelabelings.replacement**

строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.endpoints.metricRelabelings.separator**

строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.endpoints.metricRelabelings.sourceLabels**

массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**

строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.endpoints.metricRelabelings.targetLabel**

строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.endpoints.oauth2**

объект

OAuth2 для URL. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.endpoints.oauth2.clientId**

объект

Обязательный параметр

Секрет или ConfigMap, содержащие идентификатор клиента OAuth2

- **spec.endpoints.oauth2.clientId.configMap**

объект

ConfigMap, содержащий данные для использования в целях.

- **spec.endpoints.oauth2.clientId.configMap.key**
строка

Обязательный параметр

Ключ для выбора.
- **spec.endpoints.oauth2.clientId.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
- **spec.endpoints.oauth2.clientId.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.
- **spec.endpoints.oauth2.clientId.secret**
объект

Секрет, содержащий данные для использования в целях.
 - **spec.endpoints.oauth2.clientId.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.endpoints.oauth2.clientId.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.endpoints.oauth2.clientId.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.
- **spec.endpoints.oauth2.clientSecret**
объект

Обязательный параметр

Секрет, содержащий секрет клиента OAuth2
 - **spec.endpoints.oauth2.clientSecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.
 - **spec.endpoints.oauth2.clientSecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
 - **spec.endpoints.oauth2.clientSecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.
- **spec.endpoints.oauth2.endpointParams**
объект

Параметры для добавления к URL токена
- **spec.endpoints.oauth2.scopes**
массив строк

Области OAuth2, используемые для запроса токена
- **spec.endpoints.oauth2.tokenUrl**
строка

Обязательный параметр

URL для получения токена
- **spec.endpoints.params**
объект

Необязательные параметры URL HTTP
- **spec.endpoints.path**
строка

HTTP-путь для сбора метрик. Если не указано, Prometheus использует значение по умолчанию (например, `/metrics`).
- **spec.endpoints.port**
строка

Имя порта сервиса, к которому относится этот эндпоинт. Взаимоисключается с `targetPort`.
- **spec.endpoints.proxyUrl**
строка

ProxyURL, например `http://proxyserver:2195`. Направляет запросы сбора через прокси посредством этого эндпоинта.
- **spec.endpoints.relabelings**
массив объектов

RelabelConfigs для применения к выборкам перед сбором. Оператор Prometheus автоматически добавляет переименования для некоторых стандартных полей Kubernetes. Имя изначальной задачи сбора доступно через лейбл `__tmp_prometheus_job_name`. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

RelabelConfig позволяет динамически переписывать набор лейблов для целей, оповещений, собранных выборок и выборок удаленной записи. Подробнее:

https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.endpoints.relabelings.action**

строка

Действие на основе сопоставления с регулярным выражением. Действия `Uppercase` и `Lowercase` требуют Prometheus версии не ниже v2.36.0. Действия `DropEqual` и `KeepEqual` требуют Prometheus версии не ниже v2.41.0. По умолчанию: `"Replace"`

По умолчанию: `replace`

Допустимые значения: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.endpoints.relabelings.modulus**

целочисленный

Модуль для вычисления хеша значений исходных лейблов. Применимо только в случае, когда действие — `HashMod`.

- **spec.endpoints.relabelings.regex**

строка

Регулярное выражение, с которым сопоставляется извлеченное значение.

- **spec.endpoints.relabelings.replacement**

строка

Значение замены для действия `Replace`, если регулярное выражение совпадает. Доступны группы захвата из регулярного выражения.

- **spec.endpoints.relabelings.separator**

строка

Разделитель — это строка между объединенными исходными лейблами (`SourceLabels`).

- **spec.endpoints.relabelings.sourceLabels**

массив строк

Исходные лейблы выбирают значения из существующих лейблов. Их содержимое объединяется с использованием настроенного разделителя и сопоставляется с настроенным регулярным выражением.

- **Элемент массива**

строка

`LabelName` — это допустимое имя лейбла Prometheus, которое может содержать только буквы ASCII, цифры, а также символы подчеркивания.

Шаблон: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.endpoints.relabelings.targetLabel**

строка

Лейбл, в который записывается результирующая строка при замене. Она обязательна для действий `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` и `DropEqual`. Доступны группы захвата из регулярного выражения.

- **spec.endpoints.scheme**

строка

HTTP-схема для использования при сборе. Ожидаемые значения — `http` и `https`, если вы не переписываете лейбл `__scheme__` посредством переименования. Если не указано, Prometheus использует значение по умолчанию — `http`.

Допустимые значения: `http`, `https`

- **spec.endpoints.scrapeTimeout**

строка

Таймаут, после которого сбор данных прекращается. Если не указано, используется глобальный таймаут сбора Prometheus, если он не меньше `Interval`, в противном случае используется последнее.

Шаблон: `^(0|([0-9]+)y)?([0-9]+)w)?([0-9]+)d)?([0-9]+)h)?([0-9]+)m)?([0-9]+)s)?([0-9]+)ms)?$`

- **spec.endpoints.targetPort**

строка или число

Имя или номер целевого порта пода, находящегося за сервисом, порт должен быть указан со свойством порта контейнера. Взаимоисключается с `port`.

- **spec.endpoints.tlsConfig**

объект

Конфигурация TLS для использования при сборе с эндпоинта.

- **spec.endpoints.tlsConfig.ca**

объект

Удостоверяющий центр, используемый при проверке сертификатов сервера.

- **spec.endpoints.tlsConfig.ca.configMap**

объект

`ConfigMap`, содержащий данные для использования в целях.

- **spec.endpoints.tlsConfig.ca.configMap.key**

строка

Обязательный параметр

Ключ для выбора.

- **spec.endpoints.tlsConfig.ca.configMap.name**

строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.endpoints.tlsConfig.ca.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.endpoints.tlsConfig.ca.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.endpoints.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым секретным ключом.

- **spec.endpoints.tlsConfig.ca.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.endpoints.tlsConfig.ca.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.endpoints.tlsConfig.caFile**
строка

Путь к файлу сертификата удостоверяющего центра в контейнере Prometheus для целей.

- **spec.endpoints.tlsConfig.cert**
объект

Клиентский сертификат, который предъявляется для аутентификации клиента.

- **spec.endpoints.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в целях.

- **spec.endpoints.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ для выбора.

- **spec.endpoints.tlsConfig.cert.configMap.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.endpoints.tlsConfig.cert.configMap.optional**
булевый

Укажите, должен ли быть определён ConfigMap или его ключ.

- **spec.endpoints.tlsConfig.cert.secret**
объект

Секрет, содержащий данные для использования в целях.

- **spec.endpoints.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым ключом секрета.

- **spec.endpoints.tlsConfig.cert.secret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.endpoints.tlsConfig.cert.secret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.endpoints.tlsConfig.certFile**
строка

Путь к файлу клиентского сертификата в контейнере Prometheus для целей.

- **spec.endpoints.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата цели.

- **spec.endpoints.tlsConfig.keyFile**
строка

Путь к клиентскому ключевому файлу в контейнере Prometheus для целей.

- **spec.endpoints.tlsConfig.keySecret**
объект

Секрет, содержащий клиентский ключевой файл для целей.

- **spec.endpoints.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ секрета для выбора. Должен быть допустимым ключом секрета.

- **spec.endpoints.tlsConfig.keySecret.name**
строка

Имя объекта, на который ссылаются. Дополнительная информация:

<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **spec.endpoints.tlsConfig.keySecret.optional**
булевый

Укажите, должен ли быть определён Secret или его ключ.

- **spec.endpoints.tlsConfig.serverName**
строка

Используется для проверки имени хоста для целей.

- **spec.jobLabel**
строка

JobLabel выбирает лейбл из ассоциированного сервиса Kubernetes, который будет использован как лейбл `job` для всех метрик. Например: Если в `ServiceMonitor.spec.jobLabel: foo` и в `Service.metadata.labels.foo: bar`, то лейбл `job="bar"` добавляется ко всем метрикам. Если значение этого поля пустое или лейбл не существует для данного сервиса, лейбл `job` метрик по умолчанию становится именем сервиса Kubernetes.

- **spec.keepDroppedTargets**
целочисленный

Предел на количество целей, отклоненных переименованием, которые будут оставаться в памяти. 0 означает отсутствие ограничения. Требуется Prometheus $\geq$ v2.47.0.

- **spec.labelLimit**
целочисленный

Предел на количество лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelNameLengthLimit**
целочисленный

Предел на длину имени лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.labelValueLengthLimit**
целочисленный

Предел на длину значений лейблов, которые будут приниматься за одно сканирование. Действительно только в версиях Prometheus 2.27.0 и выше.

- **spec.namespaceSelector**
объект

Селектор для выбора пространств имён, из которых обнаруживаются объекты Kubernetes Endpoints.

- **spec.namespaceSelector.any**
булевый

Логическое значение, описывающее, выбраны ли все пространства имён в отличие от списка, ограничивающего их.

- **spec.namespaceSelector.matchNames**
массив строк

Список имён пространств имён для выбора.

- **spec.podTargetLabels**
массив строк

PodTargetLabels переносят лейблы на Kubernetes `Pod` на созданные метрики.

- **spec.sampleLimit**
целочисленный

SampleLimit определяет предел на количество выборок, которое будет принято за одно сканирование.

- **spec.selector**
объект

Обязательный параметр

Селектор для выбора объектов Endpoints.

- **spec.selector.matchExpressions**
массив объектов

matchExpressions — это список требований селектора лейблов. Требования объединены с помощью логического оператора И (AND).

Требование селектора лейблов — это селектор, который содержит значения, ключ и оператор, связывающий ключ и значения.

- **spec.selector.matchExpressions.key**
строка

Обязательный параметр

key — это ключ лейбла, к которому применяется селектор.

- **spec.selector.matchExpressions.operator**
строка

Обязательный параметр

operator представляет взаимоотношение ключа с набором значений. Допустимые операторы: In, NotIn, Exists и DoesNotExist.

- **spec.selector.matchExpressions.values**
массив строк

values — это массив строковых значений. Если оператор In или NotIn, массив values должен быть непустым. Если оператор Exists или DoesNotExist, массив values должен быть пустым. Этот массив заменяется во время стратегического объединения патчей.

- **spec.selector.matchLabels**
объект

matchLabels — это карта пар {ключ, значение}. Один {ключ, значение} в карте matchLabels эквивалентен элементу matchExpressions, у которого поле ключа — “key”, оператор — “In”, а массив values содержит только “value”. Требования объединены с помощью логического оператора И (AND).

- **spec.targetLabels**
массив строк

TargetLabels переносят лейблы с Kubernetes `Service` на созданные метрики.

- **spec.targetLimit**
целочисленный

TargetLimit определяет предел количества целей, которые будут приняты для сбора.

Prometheus-мониторинг

i Веб-интерфейсы, связанные с модулем: grafana

Модуль разворачивает стек мониторинга с предустановленными параметрами для Deckhouse Kubernetes Platform (DKP) и приложений, что упрощает начальную настройку.

Ключевые возможности:

- В комплекте — готовые триггеры и дашборды, поддерживается push и pull-модель сбора метрик. Нагрузка оптимизирована за счёт использования кешей и Deckhouse Prom++.
- Есть возможность хранить исторические данные с помощью даунсемплинга.
- Модуль покрывает все основные задачи базового мониторинга платформы и приложений.

Мониторинг аппаратных ресурсов

Реализовано отслеживание загрузки аппаратных ресурсов кластера с графиками по утилизации:

- процессора;
- памяти;
- диска;
- сети.

Графики доступны с агрегацией:

- по подам;
- контроллерам;
- пространствам имен;
- узлам.

Мониторинг Kubernetes

Deckhouse настраивает мониторинг параметров «здоровья» Kubernetes и таких его компонентов, как:

- общая утилизация кластера;
- связанность узлов Kubernetes между собой (измеряется rtt между всеми узлами);
- доступность и работоспособность компонентов control plane:
 - etcd;
 - coredns и kube-dns;
 - kube-apiserver и др.
- синхронизация времени на узлах и др.

Мониторинг Ingress

Подробное описание здесь

Режим расширенного мониторинга

В Deckhouse возможно использование режима расширенного мониторинга, который предоставляет алерты по дополнительным метрикам:

- свободному месту и inode на дисках узлов,
- утилизации узлов,
- доступности подов и образов контейнеров,
- истечении действия сертификатов,
- другим событиям кластера.

Алертинг в режиме расширенного мониторинга

Deckhouse предоставляет возможность гибкой настройки алертинга для каждого пространства имён и указания различной степени критичности в зависимости от порога. Можно определить множество порогов для отправки предупреждений в различные пространства имён, например, для следующих параметров:

- значения свободного места и inodes на диске;
- утилизация CPU узлов и контейнера;
- процент ошибок с кодом 5xx на ingress-nginx;
- количество возможных недоступных подов в `Deployment`, `StatefulSet`, `DaemonSet`.

Управление мониторингом как кодом (IaC подход)

Возможности системы мониторинга DKP могут быть расширены за счёт использования модуля Observability. С его помощью реализуется:

- управление алертами;
- разграничение прав доступа к настройкам и данным мониторинга;
- централизованное управление дашбордами.

Алерты

Мониторинг в составе Deckhouse включает уведомления о событиях. Стандартная поставка включает набор базовых предупреждений, охватывающих состояние кластера и его компоненты. Также остается возможность добавлять кастомные алерты.

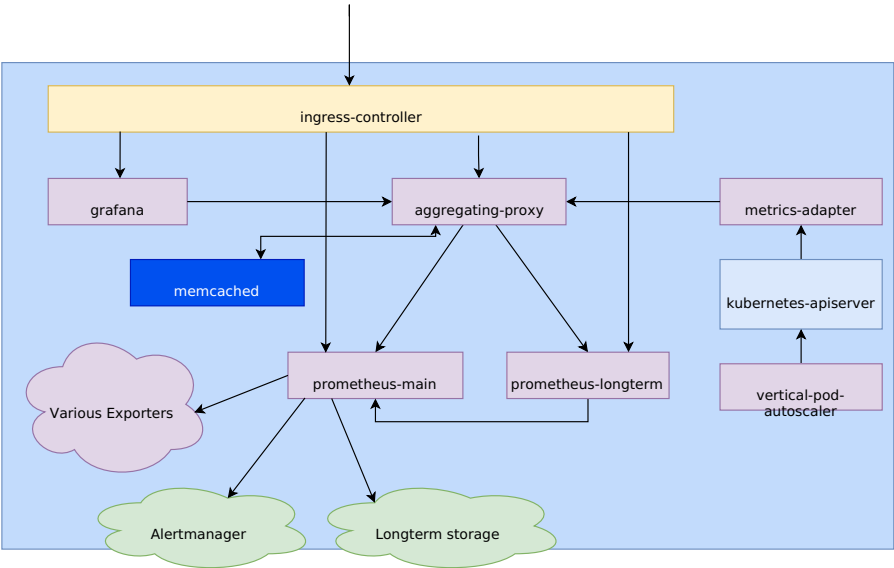
Список всех доступных алертов системы мониторинга Deckhouse Kubernetes Platform приведён на отдельной странице документации.

Отправка алертов во внешние системы

Deckhouse поддерживает отправку алертов с помощью Alertmanager:

- по протоколу SMTP;
- в PagerDuty;
- в Slack;
- в Telegram;
- посредством Webhook;
- по любым другим каналам, поддерживаемым в Alertmanager.

Архитектура



Базовые компоненты мониторинга

Компонент	Описание
prometheus-main	Основной Prometheus, который выполняет scrape каждые 30 секунд (с помощью параметра <code>scrapeInterval</code> можно изменить это значение). Он обрабатывает все правила, отправляет алерты и является основным источником данных.
prometheus-longterm	Просмотр и анализ исторических трендов.
trickster	Кеширующий прокси, снижающий нагрузку на Prometheus.
aggregating-proxy	Агрегирующий и кеширующий прокси, снижающий нагрузку на Prometheus и объединяющий main и longterm в один источник.
memcached	Сервис кеширования данных в оперативной памяти.
grafana	Управляемая платформа визуализации данных. Включает подготовленные dashboard'ы для всех модулей DKP и некоторых популярных приложений. Grafana умеет работать в режиме высокой доступности, не хранит состояние и настраивается с помощью CRD.
metrics-adapter	Компонент, соединяющий Prometheus и Kubernetes metrics API. Включает поддержку HPA в кластере Kubernetes.
vertical-pod-autoscaler	Компонент, позволяющий автоматически изменять размер запрошенных ресурсов для подов с целью оптимальной утилизации CPU и памяти.
Различные exporter'ы	Подготовленные и подключенные к Prometheus exporter'ы. Список включает множество exporter'ов для всех необходимых метрик: <code>kube-state-metrics</code> , <code>node-exporter</code> , <code>oomkill-exporter</code> , <code>image-availability-exporter</code> и многие другие.
Push/pull модель сбора метрик	Мониторинг по умолчанию использует pull-модель — данные собираются с приложений по инициативе системы мониторинга. Также поддерживается push-модель: метрики можно передавать через протокол Prometheus Remote Write или с помощью Prometheus Pushgateway.

Режим отказоустойчивости и высокой доступности мониторинга (HA)

Модуль мониторинга обеспечивает встроенную отказоустойчивость всех ключевых компонентов DKP. Все сервисы мониторинга (Prometheus-серверы, системы хранения, прокси и прочие важные компоненты) по умолчанию развертываются в нескольких копиях. Это гарантирует, что в случае сбоя отдельного экземпляра сервис продолжит работу без потери данных и доступности.

Prometheus — основной компонент сбора метрик — запускается минимум в двух копиях (при наличии достаточного количества узлов в кластере). Оба инстанса Prometheus используют одинаковую конфигурацию и получают одни и те же данные. Чтобы обеспечить бесшовную работу при отказе одной из копий для обращения к Prometheus используется специальный компонент — `aggregation-proxu`. Он позволяет объединять метрики обоих Prometheus-инстансов и всегда возвращать наиболее полные и актуальные данные, даже если одна из копий временно недоступна.

Настройки

Модуль не требует обязательной конфигурации (все работает из коробки).

В модуле реализовано 28 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left is a sidebar with navigation links: 'Обзор', 'Обновления', 'Модули' (selected), 'Глобальные настройки', and 'Kubernetes'. The main area is titled 'Система / Управление системой / Deckhouse' and 'Модули'. Below this is a table of modules:

Имя	Состояние	Этап
admission-policy-engine	включен	Gen...
backup	выключен	—
basic-auth	выключен	Gen...

On the right, the configuration page for 'admission-policy-engine' is shown. It has tabs for 'Конфигурация', 'Ресурсы', and 'YAML'. The 'Module enabled' toggle is turned on. Below it, there is a link to 'Документация к настройкам модуля admission-policy-engine'.

- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `prometheus`:

```
d8 system module enable prometheus
```

- **С помощью ModuleConfig prometheus.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `prometheus` (создайте его, при необходимости).

Пример манифеста для включения модуля `prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

The screenshot shows the 'Configuration' tab for the 'admission-policy-engine' module. It has tabs for 'Конфигурация', 'Ресурсы', and 'YAML'. The 'Module enabled' toggle is turned on. Below it, there is a link to 'Документация к настройкам модуля admission-policy-engine'. Further down, there is a section for 'Версия настроек' with a dropdown menu set to '1'.

Вы также можете отредактировать объект ModuleConfig `prometheus` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `prometheus` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc prometheus
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре

`spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig prometheus`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `operator-prometheus`: любая версия.

Конверсии

Модуль настраивается с помощью ресурса `ModuleConfig`, схема которого содержит номер версии. При применении в кластере *старой* версии схемы `ModuleConfig` выполняются автоматические преобразования. Чтобы обновить версию схемы `ModuleConfig` вручную, необходимо **последовательно** для каждой версии выполнить следующие действия:

- Обновление из версии **1** в **2**:

Удалите поле `.auth.password`. Если в результате объект `.auth` становится пуст, удалите его.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.auth**
объект

Опции, связанные с аутентификацией или авторизацией в приложении.

- **settings.auth.allowedUserEmails**
массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.allowedUserGroups**
массив строк

Массив групп, пользователям которых позволен доступ в Grafana и Prometheus.

Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.

Внимание! При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках `DexProvider`.

- **settings.auth.externalAuthentication**
объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX `external-auth`, работающий на основе модуля NGINX `auth_request`).



Внешняя аутентификация включается автоматически, если включен модуль `user-authn`.

- **settings.auth.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

Пример:

```
authSignInURL: https://example.com/dex/sign_in
```

- **settings.auth.externalAuthentication.authURL**
строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

Пример:

```
authURL: https://example.com/dex/auth
```

- **settings.auth.satisfyAny**
булевый

Разрешает пройти только одну из аутентификаций.

В комбинации с опцией `whitelistSourceRanges` позволяет считать авторизованными всех пользователей из указанных сетей без ввода логина и пароля.

По умолчанию: `false`

Пример:

```
satisfyAny: true
```

- **settings.auth.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить авторизацию в Grafana и Prometheus.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **Элемент массива**
строка

Шаблон: `^(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])|([0-9]{12}|[0-9]{3}[0-2]))?|([0-9A-Fa-f]{0,4}:){2,7}[0-9A-Fa-f]{0,4}|([0-9]|1[0-9]|0[0-9]|1[01][0-9]|12[0-8]))?)$`

- **settings.externalLabels**
объект

Набор внешних лейблов для маркировки метрик.

Допускается использование значений переменных окружения контейнера `config-reloader` в качестве значений лейблов, например:

- `HOSTNAME/POD_NAME` — содержит наименование пода (например, `prometheus-main-0`, `prometheus-main-1` и т. д.);
- `SHARD` — содержит номер шарда.

Пример:

```
externalLabels:
  prometheus_replica: "${POD_NAME}"
  shard: "${SHARD}"
  hostname: "${HOSTNAME}"
```

- **settings.grafana**
объект

Настройки для инсталляции Grafana.

- **settings.grafana.customPlugins**
массив строк

Список дополнительных plug-in'ов для Grafana.

Необходимо указать в качестве значения список имен плагинов из официального репозитория.

Пример добавления plug-in'ов для возможности указания clickhouse в качестве datasource и панели flow-chart:

```
grafana:
  customPlugins:
  - agenty-flowcharting-panel
  - vertamedia-clickhouse-datasource
```

Вы также можете устанавливать плагины из других источников передав ссылку на zip-архив плагина в формате `<url to plugin zip>;<plugin name>`:

```
grafana:
  customPlugins:
  - http://10.241.32.16:3000/netsage-bumpchart-panel-1.1.1.zip;netsage-bumpchart-panel
```

Пример:

```
customPlugins:
- agenty-flowcharting-panel
- vertamedia-clickhouse-datasource
```

- **settings.grafana.enabled**
булевый

Включить установку Grafana в кластер.

По умолчанию: `true`

Пример:

```
enabled: false
```

- **settings.grafana.useDarkTheme**
булевый

Использование по умолчанию пользовательской темной темы.

По умолчанию: `false`

Пример:

```
useDarkTheme: true
```

- **settings.highAvailability**

булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Пример:

```
highAvailability: true
```

- **settings.https**

объект

Тип сертификата, используемого для Grafana/Prometheus.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**

объект

- **settings.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для Grafana/Prometheus.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.По умолчанию: `letsencrypt`

- **settings.https.customCertificate**

объект

- **settings.https.customCertificate.secretName**

строка

Имя Secret'a в namespace `d8-system`, который будет использоваться для Grafana/Prometheus.Secret должен быть в формате `kubernetes.io/tls`.По умолчанию: `false`

- **settings.https.mode**

строка

Режим работы HTTPS:

- `Disabled` — Grafana/Prometheus будут работать только по HTTP;
- `CertManager` — Grafana/Prometheus будут работать по HTTPS и заказывать сертификат с помощью ClusterIssuer, заданного в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — Grafana/Prometheus будут работать по HTTPS, используя сертификат из namespace `d8-system`.
- `OnlyInURI` — Grafana/Prometheus будут работать по HTTP (подразумевая, что перед ними стоит внешний балансировщик, который терминирует HTTPS) и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `Disabled`Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для Grafana/Prometheus.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
ingressClass: public
```

- **settings.longtermMaxDiskSizeGigabytes**

Параметр устарел

целочисленный

Устарел и будет удален. Ни на что не влияет.

- **settings.longtermNodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
longtermNodeSelector:
  disktype: ssd
```

- **settings.longtermPodAntiAffinity**

строка

Устанавливает конфигурацию podAntiAffinity для Prometheus longterm по отношению к Prometheus main.

- **Preferred** — позволяет размещать Prometheus longterm на одном узле с Prometheus main, если нет возможности их разместить на разных узлах.
- **Required** — не позволяет размещать Prometheus longterm на одном узле с Prometheus main ни при каких условиях.

По умолчанию: **Preferred**

Допустимые значения: **Preferred**, **Required**

◦ **settings.longtermRetentionDays**

целочисленный

Сколько дней хранить данные в Longterm Prometheus. Если указать 0, Longterm Prometheus не будет запущен в кластере.

По умолчанию: **1095**

◦ **settings.longtermScrapeInterval**

строка

Как часто Longterm Prometheus должен собирать себе «снимок» данных из основного Prometheus.

По умолчанию: **5m**

Шаблон: `^([\d]*y)?([\d]*w)?([\d]*d)?([\d]*h)?([\d]*m)?([\d]*s)?$`

◦ **settings.longtermStorageClass**

строка

Имя StorageClass, который будет использован для Longterm Prometheus.

Если значение не указано, используется StorageClass существующей PVC Longterm Prometheus. Если PVC нет, будет использоваться StorageClass, согласно настройке глобального параметра storageClass.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск будет перезаказан, и все данные удалятся.

Пример:

```
longtermStorageClass: ceph-ssd
```

◦ **settings.longtermTolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
longtermTolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.longtermTolerations.effect**
строка
- **settings.longtermTolerations.key**
строка
- **settings.longtermTolerations.operator**
строка
- **settings.longtermTolerations.tolerationSeconds**
целочисленный
- **settings.longtermTolerations.value**
строка

◦ **settings.mainMaxDiskSizeGigabytes**

Параметр устарел

целочисленный

Устарел и будет удален. Ни на что не влияет.

◦ **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
nodeSelector:
  disktype: ssd
```

◦ **settings.retentionDays**

целочисленный

Сколько дней хранить данные.

По умолчанию: **15**

◦ **settings.scrapeInterval**

строка

Как часто Prometheus должен собирать метрики с target'ов.

Evaluation Interval всегда равен `scrapeInterval`.

По умолчанию: **30s**

Шаблон: `^([\d]*y)?([\d]*w)?([\d]*d)?([\d]*h)?([\d]*m)?([\d]*s)?$`

◦ **settings.storageClass**
строка

Имя StorageClass для хранения данных Prometheus.

Если не указано, используется StorageClass существующей PVC Prometheus. Если PVC нет, будет использоваться StorageClass, согласно настройке глобального параметра storageClass.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск Prometheus будет перезаказан, а все данные удалены.

Внимание. При переносе Prometheus с локальным хранилищем на другие узлы, поды зависнут в состоянии Pending. В этом случае будет необходимо сохранить базу данных Prometheus, удалить старые PVC и перезапустить под вручную. Под локальным хранилищем подразумевается StorageClass, связанный не с сетевым хранилищем, а с локальным томом на узле (например, StorageClass, создаваемый модулем local-path-provisioner).

Если указать `false`, будет принудительно использоваться `emptyDir`.

Примеры:

```
storageClass: ceph-ssd
```

```
storageClass: 'false'
```

◦ **settings.tolerations**
массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

◦ **settings.vpa**
объект

Настройки VPA для подов.

По умолчанию: `{"updateMode": "Initial"}`

Примеры:

```
vpa:
  updateMode: Initial
  longtermMaxCPU: '1'
  longtermMaxMemory: 1500Mi
  maxCPU: 1000m
  maxMemory: 1500Mi
```

```
vpa:
  updateMode: 'Off'
```

- **settings.vpa.longtermMaxCPU**
строка от число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests) для подов Longterm Prometheus.

Значение по умолчанию подбирается автоматически, исходя из максимального количества подов, которое можно создать в кластере при текущем количестве узлов и их настройках. Подробнее см. хук `detect_vpa_max` модуля.

Шаблон: `^[0-9]+m?$`

Пример:

```
longtermMaxCPU: 0.1
```

- **settings.vpa.longtermMaxMemory**
строка от число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests) для подов Longterm Prometheus.

Значение по умолчанию подбирается автоматически, исходя из максимального количества подов, которое можно создать в кластере при текущем количестве узлов и их настройках. Подробнее см. хук `detect_vpa_max` модуля.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
longtermMaxMemory: 4Mi
```

■ settings.vpa.maxCPU

строка or число

Максимальное значение, которое может выставить VPA для запроса CPU (CPU requests) для подов основного Prometheus.

Значение по умолчанию подбирается автоматически, исходя из максимального количества подов, которое можно создать в кластере при текущем количестве узлов и их настройках. Подробнее см. хук `detect_vpa_max` модуля.

Шаблон: `^[0-9]+m?$`

Пример:

```
maxCPU: '3'
```

■ settings.vpa.maxMemory

строка or число

Максимальное значение, которое может выставить VPA для запроса к памяти (memory requests) для подов основного Prometheus..

Значение по умолчанию подбирается автоматически, исходя из максимального количества подов, которое можно создать в кластере при текущем количестве узлов и их настройках. Подробнее см. хук `detect_vpa_max` модуля.

Шаблон: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

Пример:

```
maxMemory: 3Mi
```

■ settings.vpa.updateMode

строка

Режим работы VPA.

По умолчанию: `Initial`

Допустимые значения: `Initial`, `Auto`, `Off`

Аутентификация

По умолчанию используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication` (см. ниже). Если эти варианты отключены, модуль включит `basic auth` со сгенерированным паролем и пользователем `admin`.

Посмотреть сгенерированный пароль можно командой:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values prometheus -O json | jq
'.internal.auth.password'
```

Чтобы сгенерировать новый пароль, нужно удалить Secret:

```
d8 k -n d8-monitoring delete secret/basic-auth
```



Внимание. Параметр `auth.password` больше не поддерживается.

Примечание

- `retentionSize` для `main` и `longterm` **рассчитывается автоматически, возможности задать значение нет!**
 - Алгоритм расчета:
 - `pvc size * 0.85` — если PVC существует;
 - `10 GiB` — если PVC нет и `StorageClass` поддерживает ресайз;
 - `25 GiB` — если PVC нет и `StorageClass` не поддерживает ресайз.
 - Если используется `local-storage` и требуется изменить `retentionSize`, необходимо вручную изменить размер PV и PVC в нужную сторону. **Внимание!** Для расчета берется значение из `.status.capacity.storage` PVC, поскольку оно отражает реальный размер PV в случае ручного ресайза.
- `40 GiB` — размер `PersistentVolumeClaim` создаваемого по умолчанию.
- Размер дисков Prometheus можно изменить стандартным для Kubernetes способом (если в `StorageClass` это разрешено), отредактировав в `PersistentVolumeClaim` поле `.spec.resources.requests.storage`.

Custom resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

ClusterAlert

Scope: Cluster

Version: v1alpha1

Ресурс для отображения алертов Prometheus.

• metadata

объект

◦ metadata.name

строка

Идентификатор алерта (fingerprint). Соответствует идентификатору алерта в Alertmanager.

Шаблон: `^[a-z0-9]+$`

Максимальная длина: `64`

- **status**

объект

- **status.alertStatus**

строка

Current alert status.

- **status.lastUpdateTime**

строка

Timestamp of last status update for operation.

- **status.startsAt**

строка

Timestamp of alert creation.

- **alert**

объект

Обязательный параметр

Описание алерта.

- **alert.annotations**

объект

Аналогично стандартному полю `metadata.annotations`.

Пример:

```
annotations:
  summary: Certificate has expired.
```

- **alert.description**

строка

Расширенное описание алерта.

Необязательный параметр.

- **alert.labels**

объект

Обязательный параметр

Аналогично стандартному полю `metadata.labels`.

Пример:

```
labels:
  alertname: NginxIngressSslExpired
  class: ingress-nginx.deckhouse.io/nginx
```

- **alert.name**

строка

Обязательный параметр

Название алерта.

- **alert.severityLevel**

строка или число

Уровень критичности.

Необязательный параметр.

Пример:

```
severityLevel: '4'
```

- **alert.summary**

строка

Краткое описание алерта.

Необязательный параметр.

CustomAlertmanager

Scope: Cluster

Version: v1alpha1

Ресурс для подключения дополнительных Alertmanager'ов к Prometheus.

- **status**

объект

- **status.deckhouse**

объект

- **status.deckhouse.observed**

объект

Contains last timestamp when the resource change was noted by the operator and its checksum.

- **status.deckhouse.observed.checkSum**

строка

The checksum of last observed resource.

- **status.deckhouse.observed.lastTimestamp**

строка

Last timestamp when a change of the resource was noted by the operator.

- **status.deckhouse.processed**

объект

Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

- **status.deckhouse.processed.checkSum**

строка

The checksum of last processed resource.

- **status.deckhouse.processed.lastTimestamp**

строка

Last timestamp when a change of the resource was processed by the operator.

- **status.deckhouse.synced**

строка

True if last observed version of the resource was successfully applied in the cluster.

- **spec**

объект

Обязательный параметр

- **spec.external**

объект

Параметры подключения внешнего Alertmanager.

- **spec.external.address**

строка

URL-адрес внешнего сервиса Alertmanager.

Шаблон: `^https?:\/\/[^\s/$.?\#].[\s]*$`

Примеры:

```
address: http://alerts.mycompany.com
```

```
address: https://alerts.company.com
```

```
address: http://192.168.1.1
```

- **spec.external.auth**

объект

Настройки авторизации.

- **spec.external.auth.basic**

объект

Настройки Basic-авторизации.

- **spec.external.auth.basic.bearerToken**

строка

Bearer-токен.

- **spec.external.auth.basic.password**

строка

Пароль пользователя.

- **spec.external.auth.basic.username**

строка

Имя пользователя.

- **spec.external.service**

объект

Service в Kubernetes, ссылающийся на развернутый Alertmanager.

- **spec.external.service.name**

строка

Обязательный параметр

Имя Service'а Alertmanager.

- **spec.external.service.namespace**

строка

Обязательный параметр

Пространство имен Service'а Alertmanager.

- **spec.external.service.path**

строка

Префикс URL-адреса в Service для отправки алертов.

По умолчанию: `/`

- **spec.external.tls**

объект

Настройки TLS.

- **spec.external.tls.ca**

строка

CA-сертификат.

- **spec.external.tls.cert**

строка

Сертификат клиента.

- **spec.external.tls.insecureSkipVerify**
булевый

Отключение проверки сертификата.

По умолчанию: `false`

- **spec.external.tls.key**
строка

Ключ клиента.

◦ **spec.internal**
объект

Параметры настройки внутреннего Alertmanager.

- **spec.internal.inhibitRules**
массив объектов

Список правил подавления.

Правила подавления используются для подавления уведомлений от определенных оповещений, если уже сработали другие определенные оповещения (подробнее...).

- **spec.internal.inhibitRules.equal**
массив строк

Лейблы, которые должны иметь одинаковое значение в исходном и целевом алерте, чтобы подавление вступило в силу.

- **spec.internal.inhibitRules.sourceMatch**
массив объектов

Список условий, которым должен соответствовать один или более алертов, чтобы подавление вступило в силу.

Условие содержит критерии сравнения лейблов алертов.

- **spec.internal.inhibitRules.sourceMatch.name**
строка

Обязательный параметр

Имя лейбла для сравнения.

- **spec.internal.inhibitRules.sourceMatch.regex**
булевый

Тип сравнения — по равенству (`false`) или регулярному выражению (`true`).

- **spec.internal.inhibitRules.sourceMatch.value**
строка

Значение лейбла (regex-шаблон) для удовлетворения условию.

- **spec.internal.inhibitRules.targetMatch**
массив объектов

Список условий, проверяемых в алертах, при выполнении которых алерты будут подавлены.

Условие содержит критерии удовлетворяющих ему лейблов алертов.

- **spec.internal.inhibitRules.targetMatch.name**
строка

Обязательный параметр

Имя лейбла для сравнения.

- **spec.internal.inhibitRules.targetMatch.regex**
булевый

Тип сравнения — по равенству (`false`) или регулярному выражению (`true`).

- **spec.internal.inhibitRules.targetMatch.value**
строка

Значение лейбла (regex-шаблон) для удовлетворения условию.

- **spec.internal.receivers**
массив объектов

Список receiver'ов.

Receiver определяет одну или несколько интеграций для отправки оповещений.

- **spec.internal.receivers.emailConfigs**
массив объектов

Список конфигураций для отправки оповещений по электронной почте.

Конфигурация определяет параметры отправки оповещений по электронной почте.

- **spec.internal.receivers.emailConfigs.authIdentity**
строка

Идентификатор, используемый для аутентификации.

- **spec.internal.receivers.emailConfigs.authPassword**
объект

Ключ Secret'a, содержащий пароль для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.emailConfigs.authPassword.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

■ **spec.internal.receivers.emailConfigs.authPassword.name**

строка

Имя Secret'a.

■ **spec.internal.receivers.emailConfigs.authPassword.optional**

булевый

Определяет, обязательно ли (false) указанные Secret или его ключ должны существовать.

■ **spec.internal.receivers.emailConfigs.authSecret**

объект

Ключ Secret'a, содержащий CRAM-MD5-секрет для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

■ **spec.internal.receivers.emailConfigs.authSecret.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

■ **spec.internal.receivers.emailConfigs.authSecret.name**

строка

Имя Secret'a.

■ **spec.internal.receivers.emailConfigs.authSecret.optional**

булевый

Определяет, обязательно ли (false) указанные Secret или его ключ должны существовать.

■ **spec.internal.receivers.emailConfigs.authUsername**

строка

Имя пользователя, используемое для аутентификации.

■ **spec.internal.receivers.emailConfigs.from**

строка

Адрес отправителя.

■ **spec.internal.receivers.emailConfigs.headers**

массив объектов

Список дополнительных заголовков электронной почты.

Переопределяет все установленные ранее заголовки.

Определяется набором пар ключ/значение.

■ **spec.internal.receivers.emailConfigs.headers.key**

строка

Обязательный параметр

Ключ.

■ **spec.internal.receivers.emailConfigs.headers.value**

строка

Обязательный параметр

Значение.

■ **spec.internal.receivers.emailConfigs.hello**

строка

Имя хоста для идентификации на SMTP-сервере.

■ **spec.internal.receivers.emailConfigs.html**

строка

HTML-содержимое тела письма оповещения.

■ **spec.internal.receivers.emailConfigs.requireTLS**

булевый

Использовать SMTP TLS.

■ **spec.internal.receivers.emailConfigs.sendResolved**

булевый

Уведомлять или нет о том, что алерт более не активен.

■ **spec.internal.receivers.emailConfigs.smarthost**

строка

SMTP-хост, через который отправляются электронные письма.

■ **spec.internal.receivers.emailConfigs.text**

строка

Текстовое содержимое тела письма.

■ **spec.internal.receivers.emailConfigs.tlsConfig**

объект

Конфигурация TLS.

■ **spec.internal.receivers.emailConfigs.tlsConfig.ca**

объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'е.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.emailConfigs.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.key**
строка

- Обязательный параметр
- Ключ Secret'a, из которого брать данные.
- **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.name**
строка

Имя Secret'a.

 - **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

 - **spec.internal.receivers.emailConfigs.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'e.

 - **spec.internal.receivers.emailConfigs.to**
строка

Адрес e-mail, на который будут отправлены оповещения.

 - **spec.internal.receivers.name**
строка

Обязательный параметр

Имя receiver'a. Должно быть уникальным для всего списка receiver'ов.

 - **spec.internal.receivers.opsgenieConfigs**
массив объектов

Список конфигураций для отправки оповещений через OpsGenie.

Конфигурация определяет параметры отправки оповещений с помощью OpsGenie. Подробнее...

 - **spec.internal.receivers.opsgenieConfigs.apiKey**
объект

Ключ Secret'a, содержащий OpsGenie API key.

Secret должен находиться в пространстве имен `d8-monitoring`.

 - **spec.internal.receivers.opsgenieConfigs.apiKey.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

 - **spec.internal.receivers.opsgenieConfigs.apiKey.name**
строка

Имя Secret'a.

 - **spec.internal.receivers.opsgenieConfigs.apiKey.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

 - **spec.internal.receivers.opsgenieConfigs.apiURL**
строка

URL-адрес OpsGenie API для отправки запросов.

 - **spec.internal.receivers.opsgenieConfigs.description**
строка

Описание инцидента.

 - **spec.internal.receivers.opsgenieConfigs.details**
массив объектов

Список произвольных пар ключ/значение, которые содержат более подробную информацию об инциденте.

 - **spec.internal.receivers.opsgenieConfigs.details.key**
строка

Обязательный параметр

Ключ.

 - **spec.internal.receivers.opsgenieConfigs.details.value**
строка

Обязательный параметр

Значение.

 - **spec.internal.receivers.opsgenieConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.

 - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.

 - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

 - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'е.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.name**

строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'e.

- **spec.internal.receivers.opsgenieConfigs.message**
строка

Текст алерта. Максимум 130 символов.

- **spec.internal.receivers.opsgenieConfigs.note**
строка

Дополнительный текст алерта.

- **spec.internal.receivers.opsgenieConfigs.priority**
строка

Приоритет алерта. Доступны варианты P1, P2, P3, P4 и P5.

- **spec.internal.receivers.opsgenieConfigs.responders**
массив объектов

Список responder'ов, ответственных за уведомления.

При определении responder'а должно быть определено одно из полей — `id`, `name` или `username`.

- **spec.internal.receivers.opsgenieConfigs.responders.id**
строка
ID responder'а.
- **spec.internal.receivers.opsgenieConfigs.responders.name**
строка
Имя responder'а.
- **spec.internal.receivers.opsgenieConfigs.responders.type**
строка
Обязательный параметр
Тип responder'а.
- **spec.internal.receivers.opsgenieConfigs.responders.username**
строка
Имя пользователя responder'а.

- **spec.internal.receivers.opsgenieConfigs.sendResolved**
булевый

Уведомлять или нет о завершении алертов.

- **spec.internal.receivers.opsgenieConfigs.source**
строка

Обратная ссылка на отправителя уведомления.

- **spec.internal.receivers.opsgenieConfigs.tags**
строка

Разделенный запятыми список тегов, указываемый в уведомлениях.

- **spec.internal.receivers.pagerdutyConfigs**
массив объектов

Список конфигураций для отправки оповещений через PagerDuty.

Конфигурация определяет параметры отправки оповещений с помощью PagerDuty. Подробнее...

- **spec.internal.receivers.pagerdutyConfigs.class**
строка
Класс/тип события.
- **spec.internal.receivers.pagerdutyConfigs.client**
строка
Идентификатор клиента.
- **spec.internal.receivers.pagerdutyConfigs.clientURL**
строка
Обратная ссылка на отправителя уведомления.
- **spec.internal.receivers.pagerdutyConfigs.component**
строка
Компонент системы, относящийся к алерту.
- **spec.internal.receivers.pagerdutyConfigs.description**
строка
Описание инцидента.
- **spec.internal.receivers.pagerdutyConfigs.details**
массив объектов

Список произвольных пар ключ/значение, которые содержат более подробную информацию об инциденте.

- **spec.internal.receivers.pagerdutyConfigs.details.key**
строка
Обязательный параметр
Ключ.
- **spec.internal.receivers.pagerdutyConfigs.details.value**
строка
Обязательный параметр
Значение.
- **spec.internal.receivers.pagerdutyConfigs.group**
строка
Кластер или группа источников.
- **spec.internal.receivers.pagerdutyConfigs.httpConfig**
объект
Конфигурация HTTP-клиента.
 - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth**
объект
Параметры базовой аутентификации клиента.
 - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'е.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'e.

- **spec.internal.receivers.pagerdutyConfigs.routingKey**
объект

Ключ Secret'a, содержащий ключ интеграции PagerDuty (при использовании Events API v2).

Необходимо определить либо это поле, либо `serviceKey`.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.sendResolved**
булевый

Уведомлять или нет о завершении алертов.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey**
объект

Ключ Secret'a, содержащий служебный ключ PagerDuty (при использовании типа интеграции «Prometheus»).

Необходимо определить либо это поле, либо `routingKey`.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pagerdutyConfigs.severity**
строка

Уровень инцидента.

- **spec.internal.receivers.pagerdutyConfigs.url**
строка

URL-адрес для отправки запроса.

- **spec.internal.receivers.pushoverConfigs**
массив объектов

Список конфигураций для отправки оповещений через Pushover.

Конфигурация определяет параметры отправки оповещений с помощью Pushover. Подробнее

- **spec.internal.receivers.pushoverConfigs.expire**
строка

Как долго ваше уведомление будет повторяться, если пользователь не подтвердит получение уведомления.

- **spec.internal.receivers.pushoverConfigs.html**
булевый

Тип сообщения — HTML или простой текст.

- **spec.internal.receivers.pushoverConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки

подлинности.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'е.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap**

объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'е.

- **spec.internal.receivers.pushoverConfigs.message**
строка

Содержание сообщения.

- **spec.internal.receivers.pushoverConfigs.priority**
строка

Приоритет. Подробнее...

- **spec.internal.receivers.pushoverConfigs.retry**
строка

Как часто серверы Pushover будут отправлять одно и то же уведомление пользователю. Должно быть не менее 30 секунд.

- **spec.internal.receivers.pushoverConfigs.sendResolved**
булевый

Уведомлять или нет о завершении алертов.

- **spec.internal.receivers.pushoverConfigs.sound**
строка

Переопределение звука по умолчанию.

- **spec.internal.receivers.pushoverConfigs.title**
строка

Заголовок уведомления.

- **spec.internal.receivers.pushoverConfigs.token**
объект

Ключ Secret'a, содержащий API-токен зарегистрированного приложения (подробнее...).

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pushoverConfigs.token.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.token.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.token.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.pushoverConfigs.url**
строка

Дополнительный URL-адрес, отображаемый рядом с сообщением.

- **spec.internal.receivers.pushoverConfigs.urlTitle**
строка

Заголовок для дополнительного URL-адреса, при его отсутствии отображается только URL-адрес.

- **spec.internal.receivers.pushoverConfigs.userKey**
объект

Ключ Secret'a, содержащий пользовательский ключ пользователя-получателя.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.pushoverConfigs.userKey.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.pushoverConfigs.userKey.name**
строка

Имя Secret'a.

- **spec.internal.receivers.pushoverConfigs.userKey.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.slackConfigs**
массив объектов

Список конфигураций для отправки оповещений через Slack.

Конфигурация определяет параметры отправки оповещений с помощью Slack. Подробнее...

- **spec.internal.receivers.slackConfigs.actions**
массив объектов

Список Slack actions, которые отправляются с каждым уведомлением.

Настраивает один Slack action, который отправляется с каждым уведомлением.

Больше информации здесь и здесь.

- **spec.internal.receivers.slackConfigs.actions.confirm**
объект

Защищает пользователей от деструктивных действий, прося их подтвердить нажатие кнопки еще раз. Подробнее...

- **spec.internal.receivers.slackConfigs.actions.confirm.dismissText**
строка

Надпись на кнопке отмены.

- **spec.internal.receivers.slackConfigs.actions.confirm.okText**
строка

Надпись на кнопке подтверждения.

- **spec.internal.receivers.slackConfigs.actions.confirm.text**
строка

Обязательный параметр

Подробное описание последствий выполнения действия и назначения кнопок.

Не более 30 символов.

- **spec.internal.receivers.slackConfigs.actions.confirm.title**
строка

Короткий заголовок всплывающего окна.

- **spec.internal.receivers.slackConfigs.actions.name**
строка

- **spec.internal.receivers.slackConfigs.actions.style**
строка

- **spec.internal.receivers.slackConfigs.actions.text**
строка

Обязательный параметр

- **spec.internal.receivers.slackConfigs.actions.type**
строка

Обязательный параметр
- **spec.internal.receivers.slackConfigs.actions.url**
строка
- **spec.internal.receivers.slackConfigs.actions.value**
строка
- **spec.internal.receivers.slackConfigs.apiUrl**
объект

Ключ Secret'a, содержащий URL-адрес Slack webhook.

Secret должен находиться в пространстве имен `d8-monitoring`.
 - **spec.internal.receivers.slackConfigs.apiUrl.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.
 - **spec.internal.receivers.slackConfigs.apiUrl.name**
строка

Имя Secret'a.
 - **spec.internal.receivers.slackConfigs.apiUrl.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.
- **spec.internal.receivers.slackConfigs.callbackId**
строка
- **spec.internal.receivers.slackConfigs.channel**
строка

Канал или пользователь для отправки уведомлений.
- **spec.internal.receivers.slackConfigs.color**
строка
- **spec.internal.receivers.slackConfigs.fallback**
строка
- **spec.internal.receivers.slackConfigs.fields**
массив объектов

Список полей Slack, которые отправляются с каждым уведомлением.

Каждый элемент списка настраивает одно поле Slack, которое отправляется с каждым уведомлением.

Каждое поле должно содержать заголовок (`title`), значение (`value`) и необязательный параметр `short`, указывающий, достаточно ли короткое поле для отображения рядом с другими короткими полями.
Подробнее...
 - **spec.internal.receivers.slackConfigs.fields.short**
булевый
 - **spec.internal.receivers.slackConfigs.fields.title**
строка

Обязательный параметр
 - **spec.internal.receivers.slackConfigs.fields.value**
строка

Обязательный параметр
- **spec.internal.receivers.slackConfigs.footer**
строка
- **spec.internal.receivers.slackConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.
 - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.slackConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'e.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

Конфигурация определяет параметры отправки оповещений с помощью Telegram.

Подробнее см. в документации Prometheus...

■ **spec.internal.receivers.telegramConfigs.apiUrl**
строка

Адрес Telegram API, например `https://api.telegram.org`.

Если не указано, будет использоваться адрес API по умолчанию.

■ **spec.internal.receivers.telegramConfigs.botToken**
объект

Ключ Secret'a, содержащий токен Telegram-бота.

Secret должен находиться в пространстве имен `d8-monitoring`.

■ **spec.internal.receivers.telegramConfigs.botToken.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

■ **spec.internal.receivers.telegramConfigs.botToken.name**
строка

Имя Secret'a.

■ **spec.internal.receivers.telegramConfigs.botToken.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

■ **spec.internal.receivers.telegramConfigs.chatID**
целочисленный

Идентификатор чата Telegram.

■ **spec.internal.receivers.telegramConfigs.disableNotifications**
булевый

Отключить/включить уведомления Telegram.

■ **spec.internal.receivers.telegramConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization**
объект

Конфигурация заголовка авторизации для клиента.

Это взаимоисключающая опция с разделом BasicAuth.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials**
объект

Secret, содержащий полномочия для выполнения запроса.

Secret должен находиться в пространстве имен `d8-monitoring`.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.name**
строка

Имя Secret'a.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

■ **spec.internal.receivers.telegramConfigs.httpConfig.authorization.type**
строка

Устанавливает тип аутентификации.

По умолчанию — Bearer. Basic будет вызывать ошибку.

■ **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.

Это взаимоисключающая опция с разделом Authorization. Если оба определены, BasicAuth имеет приоритет.

■ **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password**
объект

Secret, который содержит пароль для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

■ **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.name**
строка
Имя Secret'a.
- **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.optional**
булевый
Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.
- **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username**
объект
Secret, который содержит имя пользователя для аутентификации.
Secret должен находиться в пространстве имен `d8-monitoring`.
 - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.key**
строка
Обязательный параметр
Ключ Secret'a, из которого брать данные.
 - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.name**
строка
Имя Secret'a.
 - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.optional**
булевый
Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.
- **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret**
объект
Secret, содержащий токен носителя, который будет использоваться клиентом для аутентификации.
Secret должен находиться в пространстве имен `d8-monitoring`.
 - **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.key**
строка
Обязательный параметр
Ключ Secret'a, из которого брать данные.
 - **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.name**
строка
Имя Secret'a.
 - **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.optional**
булевый
Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.
- **spec.internal.receivers.telegramConfigs.httpConfig.followRedirects**
булевый
Указывает, должен ли клиент следовать перенаправлениям HTTP 3xx.
- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2**
объект
Учетные данные клиента OAuth2, используемые для получения токена target'a.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId**
объект
Обязательный параметр
Secret или ConfigMap, содержащие идентификатор клиента OAuth2.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap**
объект
ConfigMap, который содержит данные, используемые для target'a.
ConfigMap должен находиться в пространстве имен `d8-monitoring`.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.key**
строка
Обязательный параметр
Ключ ConfigMap, из которого брать данные.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.name**
строка
Имя ConfigMap.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.optional**
булевый
Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret**
объект
Secret, который содержит данные, используемые в target'e.
Secret должен находиться в пространстве имен `d8-monitoring`.
 - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret**
объект

Обязательный параметр

Secret, который содержит клиентские данные OAuth2.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.endpointParams**
объект

Параметры для добавления к URL-адресу токена.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.scopes**
массив строк

Области действия OAuth2, используемые для запроса токена.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.tokenUrl**
строка

Обязательный параметр

URL-адрес для получения токена.

- **spec.internal.receivers.telegramConfigs.httpConfig.proxyURL**
строка

Опциональный URL-адрес прокси.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS для клиента.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca**
объект

Центр сертификации, используемый при проверке сертификатов сервера.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, который содержит данные для использования в target'e.

ConfigMap должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, который содержит данные для использования в target'e.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert**
объект

Сертификат клиента для представления при выполнении аутентификации клиента.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, который содержит данные для использования в target'e.

ConfigMap должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, который содержит данные для использования в target'e.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключает проверку целевого сертификата.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, который содержит файл клиентского ключа target'a.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.serverName**
строка

Используется для проверки имени хоста для target'a.

- **spec.internal.receivers.telegramConfigs.message**
строка

Шаблон сообщения

- **spec.internal.receivers.telegramConfigs.parseMode**
строка

Режим разбора сообщения Telegram.

Допустимые значения: `MarkdownV2`, `Markdown`, `HTML`

- **spec.internal.receivers.telegramConfigs.sendResolved**
булевый

Уведомлять ли о урегулированных предупреждениях.

- **spec.internal.receivers.victoropsConfigs**
массив объектов

Список конфигураций для отправки оповещений через VictorOps.

Конфигурация определяет параметры отправки оповещений с помощью VictorOps. Подробнее...

- **spec.internal.receivers.victoropsConfigs.apiKey**
объект

Ключ Secret'a, содержащий ключ API для использования при обращении к API VictorOps.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.victoropsConfigs.apiKey.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.apiKey.name**
строка

Имя Secret'a.

- **spec.internal.receivers.victoropsConfigs.apiKey.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.apiUrl**
строка

URL-адрес VictorOps API.

- **spec.internal.receivers.victoropsConfigs.customFields**
массив объектов

Список дополнительных настраиваемых полей для уведомлений.

Определяется набором пар ключ/значение.

- **spec.internal.receivers.victoropsConfigs.customFields.key**
строка

Обязательный параметр

Ключ.

- **spec.internal.receivers.victoropsConfigs.customFields.value**
строка

Обязательный параметр

Значение.

- **spec.internal.receivers.victoropsConfigs.entityDisplayName**
строка

Краткое описание проблемы.

- **spec.internal.receivers.victoropsConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'e.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth**

объект

Параметры базовой аутентификации клиента.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password**

объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.name**

строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.optional**

булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username**

объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.name**

строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.optional**

булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret**

объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.key**

строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.name**

строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.optional**

булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.proxyURL**

строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig**

объект

Конфигурация TLS.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca**

объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap**

объект

ConfigMap, содержащий данные для использования в target'е.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.key**

строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.name**

строка

Имя ConfigMap.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.optional**

булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны

существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'e.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'e.

- **spec.internal.receivers.webhookConfigs.maxAlerts**
целочисленный

Максимальное количество алертов, отправляемых на одно сообщение webhook.

Если 0 — в сообщение включаются все алерты.

Допустимые значения: 0 <= X

- **spec.internal.receivers.webhookConfigs.sendResolved**
булевый

Уведомлять или нет о завершении алертов.

- **spec.internal.receivers.webhookConfigs.url**
строка

URL-адрес для отправки HTTP POST-запросов.

`urlSecret` имеет приоритет над `url`.

Должен быть определен либо `urlSecret`, либо `url`.

- **spec.internal.receivers.webhookConfigs.urlSecret**
объект

Ключ Secret'a, содержащий URL-адрес webhook для отправки HTTP-запросов.

`urlSecret` имеет приоритет над `url`.

Должен быть определен либо `urlSecret`, либо `url`.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.webhookConfigs.urlSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.webhookConfigs.urlSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.webhookConfigs.urlSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs**
массив объектов

Список конфигураций для отправки оповещений через WeChat.

Конфигурация определяет параметры отправки оповещений с помощью WeChat. Подробнее...

- **spec.internal.receivers.wechatConfigs.agentID**
строка

- **spec.internal.receivers.wechatConfigs.apiSecret**
объект

Ключ Secret'a, содержащий ключ API WeChat.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.wechatConfigs.apiSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.apiSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.apiSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.apiUrl**
строка

URL-адрес WeChat API.

- **spec.internal.receivers.wechatConfigs.corpID**
строка

Идентификатор корпорации для аутентификации.

- **spec.internal.receivers.wechatConfigs.httpConfig**
объект

Конфигурация HTTP-клиента.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth**
объект

Параметры базовой аутентификации клиента.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password**
объект

Secret в пространстве имен `d8-monitoring`, который содержит пароль для проверки подлинности.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username**
объект

Secret в пространстве имен `d8-monitoring`, который содержит имя пользователя для проверки подлинности.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret**
объект

Ключ Secret'a, содержащий bearer-токен, который будет использоваться клиентом для аутентификации.

Secret должен находиться в пространстве имен `d8-monitoring`.

- **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.proxyURL**
строка

URL-адрес прокси-сервера (опционально).

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig**
объект

Конфигурация TLS.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca**
объект

Структура, содержащая сертификат CA для target'ов.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret**
объект

Secret, содержащий данные для использования в target'e.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.name**

строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert**
объект

Структура, содержащая файл клиентского сертификата target'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap**
объект

ConfigMap, содержащий данные для использования в target'e.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.key**
строка

Обязательный параметр

Ключ ConfigMap, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.name**
строка

Имя ConfigMap.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.optional**
булевый

Определяет, обязательно ли (`false`) указанные ConfigMap или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret**
объект

Secret, содержащий данные, используемые для target'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.insecureSkipVerify**
булевый

Отключение проверки сертификатов target'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret**
объект

Secret, содержащий файл ключа клиента для target'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.key**
строка

Обязательный параметр

Ключ Secret'a, из которого брать данные.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.name**
строка

Имя Secret'a.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.optional**
булевый

Определяет, обязательно ли (`false`) указанные Secret или его ключ должны существовать.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.serverName**
строка

Имя сервера, используемое для проверки на target'e.

- **spec.internal.receivers.wechatConfigs.message**
строка

Данные для запроса через API WeChat. API.

- **spec.internal.receivers.wechatConfigs.messageType**
строка

- **spec.internal.receivers.wechatConfigs.sendResolved**
булевый

Уведомлять или нет о завершении алертов.

- **spec.internal.receivers.wechatConfigs.toParty**
строка

- **spec.internal.receivers.wechatConfigs.toTag**
строка

- **spec.internal.receivers.wechatConfigs.toUser**
строка
 - **spec.internal.route**
объект
- Обязательный параметр
- Определение маршрута алертов, соответствующих пространству имен ресурса.
- Если он присутствует, то будет добавлен в сгенерированную конфигурацию Alertmanager'а в качестве маршрута первого уровня.
- **spec.internal.route.continue**
булевый
- Продолжать ли проверять условие алерта к дочерним узлам.
- Оператор Prometheus всегда будет переопределять значение на true для маршрута первого уровня.
- **spec.internal.route.groupBy**
массив строк
- Список лейблов для группировки.
- **spec.internal.route.groupInterval**
строка
- Время ожидания, прежде чем отправлять обновленное уведомление.
- Должно соответствовать регулярному выражению `[0-9]+(ms|s|m|h)` (миллисекунды, секунды, минуты, часы).
- **spec.internal.route.groupWait**
строка
- Время ожидания, прежде чем отправлять первоначальное уведомление.
- Должно соответствовать регулярному выражению `[0-9]+(ms|s|m|h)` (миллисекунды, секунды, минуты, часы).
- **spec.internal.route.matchers**
массив объектов
- Список условий, которым должны соответствовать лейблы алертов.
- Для маршрута первого уровня оператор удаляет все существующие совпадения равенства и регулярных выражений на лейбле `namespace` и добавляет совпадения `namespace: <object namespace>`.
- Условие содержит критерии удовлетворяющих ему лейблов алертов.
- **spec.internal.route.matchers.name**
строка
 - Обязательный параметр
 - Имя лейбла для сравнения.
 - **spec.internal.route.matchers.regex**
булевый
 - Тип сравнения — по равенству (false) или регулярному выражению (true).
 - **spec.internal.route.matchers.value**
строка
 - Значение лейбла (regex-шаблон) для удовлетворения условию.
- **spec.internal.route.receiver**
строка
- Имя receiver'а для этого маршрута.
- Если он не пустой, то должен быть указан в поле `receivers`.
- **spec.internal.route.repeatInterval**
строка
- Время ожидания, прежде чем повторять последнее уведомление.
- Должно соответствовать регулярному выражению `[0-9]+(ms|s|m|h)` (миллисекунды, секунды, минуты, часы).
- **spec.internal.route.routes**
массив
- Дочерние маршруты.
- **spec.type**
строка
- Тип Alertmanager'а.
- Допустимые значения: `External`, `Internal`

CustomPrometheusRules

Scope: Cluster

- v1
- v1alpha1

Ресурс хранения `PrometheusRule`.

Подробнее....

- **spec**
объект
- Обязательный параметр
- **spec.groups**
массив объектов

Обязательный параметр

Описание группы алертов.

Структура групп полностью совпадает с аналогичной в prometheus-operator.

Список последовательно применяемых recording- и alerting-правил.

- **spec.groups.interval**

строка

- **spec.groups.name**

строка

Обязательный параметр

- **spec.groups.rules**

массив объектов

Обязательный параметр

- **spec.groups.rules.alert**

строка

- **spec.groups.rules.annotations**

объект

- **spec.groups.rules.expr**

строка или число

Обязательный параметр

- **spec.groups.rules.for**

строка

- **spec.groups.rules.labels**

объект

- **spec.groups.rules.record**

строка

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Ресурс хранения `PrometheusRule`.

Подробнее....

- **spec**
объект

Обязательный параметр

- **spec.groups**
массив объектов

Обязательный параметр

Описание группы алертов.

Структура групп полностью совпадает с аналогичной в prometheus-operator.

Список последовательно применяемых recording- и alerting-правил.

- **spec.groups.interval**

строка

- **spec.groups.name**

строка

Обязательный параметр

- **spec.groups.rules**

массив объектов

Обязательный параметр

- **spec.groups.rules.alert**

строка

- **spec.groups.rules.annotations**

объект

- **spec.groups.rules.expr**

строка или число

Обязательный параметр

- **spec.groups.rules.for**

строка

- **spec.groups.rules.labels**

объект

- **spec.groups.rules.record**

строка

GrafanaAdditionalDatasource

Scope: Cluster

- v1
- v1alpha1

Ресурс для подключения дополнительных datasource к Grafana.

Параметры ресурса подробно описаны в документации к Grafana.

- **spec**
объект

Обязательный параметр

- **spec.access**
строка

Обязательный параметр

Режим доступа.

Допустимые значения: `Proxy`, `Direct`

- **spec.basicAuth**
булевый

Включить/выключить использование базовой авторизации.

- **spec.basicAuthUser**
строка

Имя пользователя для базовой авторизации.

- **spec.database**
строка

Имя БД (если используется).

- **spec.jsonData**
объект

Поля, которые будут конвертированы в JSON и сохранены в jsonData.

- **spec.secureJsonData**
объект

JSON-объект данных, который будет сохранен зашифрованным.

- **spec.type**
строка

Обязательный параметр

Тип datasource.

- **spec.url**
строка

URL-адрес datasource.

Пример:

```
url: http://localhost:8080
```

- **spec.user**
строка

Имя пользователя БД.

- **spec.withCredentials**
булевый

Включение/отключение флага withCredentials (влияет на формирование заголовков).

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Ресурс для подключения дополнительных datasource к Grafana.

Параметры ресурса подробно описаны в документации к Grafana.

- **spec**
объект

Обязательный параметр

- **spec.access**
строка

Обязательный параметр

Режим доступа.

Допустимые значения: `proxy`, `direct`

- **spec.basicAuth**
булевый

Включить/выключить использование базовой авторизации.

- **spec.basicAuthUser**
строка

Имя пользователя для базовой авторизации.

- **spec.database**
строка

Имя БД (если используется).

- **spec.jsonData**
объект

Поля, которые будут конвертированы в JSON и сохранены в jsonData.

- **spec.secureJsonData**
объект

JSON-объект данных, который будет сохранен зашифрованным.

- **spec.type**
строка

Обязательный параметр

Тип datasource.

- **spec.url**
строка

URL-адрес datasource.

Пример:

```
url: http://localhost:8080
```

- **spec.user**

строка

Имя пользователя БД.

- **spec.withCredentials**

булевый

Включение/отключение флага withCredentials (влияет на формирование заголовков).

GrafanaAlertsChannel

Scope: Cluster

Version: v1alpha1

Ресурс для добавления каналов оповещения в Grafana.

- **spec**

объект

Обязательный параметр

- **spec.alertManager**

объект

Обязательный параметр

Настройка канала внешнего Prometheus Alert Manager.

- **spec.alertManager.address**

строка

Обязательный параметр

URL-адрес внешнего Alertmanager.

Шаблон: `^https?:/[^\s/$. ?#]. [^\s]*$`

Примеры:

```
address: http://alerts.mycompany.com
```

```
address: https://alerts.company.com
```

```
address: http://192.168.1.1
```

- **spec.alertManager.auth**

объект

Параметры авторизации.

- **spec.alertManager.auth.basic**

объект

Обязательный параметр

Параметры базовой авторизации.

- **spec.alertManager.auth.basic.password**

строка

Обязательный параметр

Пароль.

- **spec.alertManager.auth.basic.username**

строка

Обязательный параметр

Имя пользователя.

- **spec.description**

строка

Небольшое необязательное описание для этого канала предупреждений.

По умолчанию:

- **spec.disableResolveMessage**

булевый

Отключение отправки сообщения о переходе алерта в неактивное состояние.

По умолчанию: `false`

- **spec.isDefault**

булевый

Использование этого канала для всех алертов.

По умолчанию: `false`

- **spec.type**

строка

Обязательный параметр

Допустимые значения: `PrometheusAlertManager`

GrafanaDashboardDefinition

Scope: Cluster

- v1
- v1alpha1

Ресурс хранения и управления dashboard в Grafana.

- **spec**
объект

Обязательный параметр

- **spec.definition**
строка

Обязательный параметр

JSON-манифест dashboard.

i **Важно!** Следите, чтобы, помимо `uid`, в манифесте не было «местного» `id` по адресу `.id`.

Шаблон: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

- **spec.folder**
строка

Обязательный параметр

Папка, в которую попадет данный dashboard. Если такой папки нет, она будет создана.

Пример:

```
folder: My Folder
```

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Ресурс хранения и управления dashboard в Grafana.

- **spec**
объект

Обязательный параметр

- **spec.definition**
строка

Обязательный параметр

JSON-манифест dashboard.

i **Важно!** Следите, чтобы, помимо `uid`, в манифесте не было «местного» `id` по адресу `.id`.

Шаблон: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

- **spec.folder**
строка

Обязательный параметр

Папка, в которую попадет данный dashboard. Если такой папки нет, она будет создана.

Пример:

```
folder: My Folder
```

PrometheusRemoteWrite

Scope: Cluster

- v1
- v1alpha1

Ресурс для включения `remote_write` данных из локального Prometheus в отдельный longterm storage (например, VictoriaMetrics).

Таких ресурсов в кластере может быть любое количество.

- **spec**
объект

Обязательный параметр

- **spec.basicAuth**
объект

Параметры базовой авторизации для отправки данных.

- **spec.basicAuth.password**
строка

Обязательный параметр

Пароль для аутентификации.

- **spec.basicAuth.username**
строка

Обязательный параметр

Имя пользователя для аутентификации.

- **spec.bearerToken**

строка

Bearer-токен.

- **spec.customAuthToken**

строка

Пользовательский токен, передаваемый в качестве заголовка `X-Auth-Token`.

- **spec.headers**

объект

HTTP-заголовки, добавляемые к запросу.

- **spec.headers.X-Scope-OrgID**

строка

Обязательный параметр

Заголовок `X-Scope-OrgID`, указывающий ID тенанта.

- **spec.tlsConfig**

объект

Параметры TLS.

- **spec.tlsConfig.ca**

строка

Сертификат ЦС (CA), используемый при проверке сертификата API-сервера.

По умолчанию:

- **spec.tlsConfig.insecureSkipVerify**

булевый

Отключить проверку сертификата.

По умолчанию: `false`

- **spec.url**

строка

Обязательный параметр

Адрес, по которому Prometheus будет отправлять данные.

Пример:

```
url: https://victoriametrics-test.domain.com/api/v1/write
```

- **spec.writeRelabelConfigs**

массив объектов

Параметры для relabel'инга данных для отправки.

Например, если необходимо удалить лишние метрики или произвести релейбл данных.

- **spec.writeRelabelConfigs.action**

строка

Действие, выполняемое при соответствии регулярному выражению.

По умолчанию: `replace`

- **spec.writeRelabelConfigs.modulus**

целочисленный

Модуль для хеширования значений исходного лейбла.

- **spec.writeRelabelConfigs.regex**

строка

Регулярное выражение для применения к извлеченному значению.

По умолчанию: `(.*)`

- **spec.writeRelabelConfigs.replacement**

строка

На что заменять в случае соответствия шаблону регулярного выражения.

Доступны regex-группы.

По умолчанию: `$1`

- **spec.writeRelabelConfigs.separator**

строка

Символ, разделяющий исходные лейблы.

По умолчанию: `;`

- **spec.writeRelabelConfigs.sourceLabels**

массив строк

Исходные лейблы для выбора значений.

Исходные лейблы объединяются с учетом разделителя (`separator`), и к ним применяется фильтр регулярного выражения, по результатам чего содержимое заменяется, принимается или удаляется.

- **spec.writeRelabelConfigs.targetLabel**

строка

Лейбл, в который записывается результирующее значение при замене.

Поле является обязательным при `action: replace`.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Ресурс для включения `remote_write` данных из локального Prometheus в отдельный longterm storage (например, VictoriaMetrics).

Таких ресурсов в кластере может быть любое количество.

- **spec**
объект

Обязательный параметр

- **spec.basicAuth**
объект

Параметры базовой авторизации для отправки данных.

- **spec.basicAuth.password**
строка

Обязательный параметр

Пароль для аутентификации.

- **spec.basicAuth.username**
строка

Обязательный параметр

Имя пользователя для аутентификации.

- **spec.bearerToken**
строка

Bearer-токен.

- **spec.customAuthToken**
строка

Пользовательский токен, передаваемый в качестве заголовка `X-Auth-Token`.

- **spec.headers**
объект

HTTP-заголовки, добавляемые к запросу.

- **spec.headers.X-Scope-OrgID**
строка

Обязательный параметр

Заголовок `X-Scope-OrgID`, указывающий ID тенанта.

- **spec.tlsConfig**
объект

Параметры TLS.

- **spec.tlsConfig.ca**
строка

Сертификат ЦС (CA), используемый при проверке сертификата API-сервера.

По умолчанию:

- **spec.tlsConfig.insecureSkipVerify**
булевый

Отключить проверку сертификата.

По умолчанию: `false`

- **spec.url**
строка

Обязательный параметр

Адрес, по которому Prometheus будет отправлять данные.

Пример:

```
url: https://victoriametrics-test.domain.com/api/v1/write
```

- **spec.writeRelabelConfigs**
массив объектов

Параметры для relabel'инга данных для отправки.

Например, если необходимо удалить лишние метрики или произвести релейбл данных.

- **spec.writeRelabelConfigs.action**
строка

Действие, выполняемое при соответствии регулярному выражению.

По умолчанию: `replace`

- **spec.writeRelabelConfigs.modulus**
целочисленный

Модуль для хеширования значений исходного лейбла.

- **spec.writeRelabelConfigs.regex**
строка

Регулярное выражение для применения к извлеченному значению.

По умолчанию: `(.*)`

- **spec.writeRelabelConfigs.replacement**

строка

На что заменять в случае соответствия шаблону регулярного выражения.

Доступны регехр-группы.

По умолчанию: `$1`

■ **spec.writeRelabelConfigs.separator**

строка

Символ, разделяющий исходные лейблы.

По умолчанию: `;`

■ **spec.writeRelabelConfigs.sourceLabels**

массив строк

Исходные лейблы для выбора значений.

Исходные лейблы объединяются с учетом разделителя (`separator`), и к ним применяется фильтр регулярного выражения, по результатам чего содержимое заменяется, принимается или удаляется.

■ **spec.writeRelabelConfigs.targetLabel**

строка

Лейбл, в который записывается результирующее значение при замене.

Поле является обязательным при `action: replace`.

Примеры конфигурации

Пример конфигурации модуля

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings:
    auth:
      password: xxxxxx
    retentionDays: 7
    storageClass: rbd
    nodeSelector:
      node-role/monitoring: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: monitoring

```

Запись данных Prometheus в longterm storage

У Prometheus есть поддержка `remote_write` данных из локального Prometheus в отдельный longterm storage (например, VictoriaMetrics). В Deckhouse поддержка этого механизма реализована с помощью кастомного ресурса `PrometheusRemoteWrite`.



Для VictoriaMetrics подробную информацию о способах передачи данные в vmaagent можно получить в документации VictoriaMetrics.

Пример минимального PrometheusRemoteWrite

```

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write

```

Пример расширенного PrometheusRemoteWrite

```

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
  basicAuth:
    username: username
    password: password
  writeRelabelConfigs:
    - sourceLabels: [__name__]
      action: keep
      regex: prometheus_build_.*|my_cool_app_metrics_.*
    - sourceLabels: [__name__]
      action: drop
      regex: my_cool_app_metrics_with_sensitive_data

```

Подключение Prometheus к сторонней Grafana

У каждого `ingress-nginx-controller` есть сертификаты, при указании которых в качестве клиентских будет разрешено подключение к Prometheus. Все, что нужно, — создать дополнительный Ingress-ресурс.

i В приведенном ниже примере предполагается, что Secret `example-com-tls` уже существует в namespace `d8-monitoring`.
Имена для Ingress `my-prometheus-api` и Secret `my-basic-auth-secret` указаны для примера. Замените их на более подходящие для вас.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-prometheus-api
  namespace: d8-monitoring
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: my-basic-auth-secret
    nginx.ingress.kubernetes.io/app-root: /graph
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_ssl_certificate /etc/nginx/ssl/client.crt;
      proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
      proxy_ssl_protocols TLSv1.2;
      proxy_ssl_session_reuse on;
spec:
  ingressClassName: nginx
  rules:
  - host: prometheus-api.example.com
    http:
      paths:
      - backend:
          service:
            name: prometheus
            port:
              name: https
          path: /
          pathType: ImplementationSpecific
    tls:
      - hosts:
          - prometheus-api.example.com
        secretName: example-com-tls
---
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-auth-secret
  namespace: d8-monitoring
type: Opaque
data:
  # Строка basic-auth хешируется с помощью htpasswd.
  auth: Zm9vOiRhchIXJE9GRzNYeWJwJGNrTDBGSERBa29YWUlsSDkuY3ZlVDAK # foo:bar
```

Добавьте data source в Grafana:

i В качестве URL необходимо указать `https://prometheus-api.<домен-вашего-кластера>*`










Data Sources / FlantPrometheus

Type: Prometheus

Settings

Dashboards

Name

FlantPrometheus

Default

☐

HTTP

URL

https://prometheus-api.<домен-вашего-...>

Whitelisted Cookies

Add Name

Add

Auth

Basic auth

☒

With Credentials

☐

TLS Client Auth

☐

With CA Cert

☐

Skip TLS Verify

☐

Forward OAuth Identity

☐

Basic Auth Details

User

foo

Password

configured

reset

Scrape interval

Query timeout

60s

HTTP Method

Choose

- **Basic-авторизация** не является надежной мерой безопасности. Рекомендуется ввести дополнительные меры безопасности, например указать аннотацию `nginx.ingress.kubernetes.io/whitelist-source-range`.
- Из-за необходимости создания Ingress-ресурса в системном пространстве имён подключение таким способом **не рекомендуется**. Deckhouse **не гарантирует** сохранение работоспособности данной схемы подключения в связи с его активными постоянными обновлениями.
- Этот Ingress-ресурс может быть использован для доступа к Prometheus API не только для Grafana, но и для других интеграций, например, для федерации Prometheus.

Подключение стороннего приложения к Prometheus

Подключение к Prometheus защищено с помощью kube-rbac-proxy. Для подключения создайте ServiceAccount с необходимыми правами.

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app:prometheus-access
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["prometheuses/http"]
  resourceNames: ["main", "longterm"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app:prometheus-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app:prometheus-access
subjects:
- kind: ServiceAccount
  name: app
  namespace: default

```

Выполните запрос, используя команду `curl`:

```

apiVersion: batch/v1
kind: Job
metadata:
  name: app-curl
  namespace: default
spec:
  template:
    metadata:
      name: app-curl
    spec:
      serviceAccountName: app
      containers:
      - name: app-curl
        image: curlimages/curl:7.69.1
        command: ["sh", "-c"]
        args:
        - >-
          curl -H "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" -k -f
          https://prometheus.d8-monitoring:9090/api/v1/query_range?query=up\&start=1584001500\&end=1584023100\&step=30
        restartPolicy: Never
      backoffLimit: 4

```

`Job` должен завершиться успешно.

Отправка алертов в Telegram

Alertmanager поддерживает прямую отправку алертов в Telegram.

Создайте Secret в пространстве имен `d8-monitoring`:

```

apiVersion: v1
kind: Secret
metadata:
  name: telegram-bot-secret
  namespace: d8-monitoring
stringData:
  token: "562696849:AAExcuJ8H6z4pTlPuocbrXXXXXXXXXXXX"

```

Разверните кастомный ресурс `CustomAlertManager`:


```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: telegram
spec:
  type: Internal
  internal:
    receivers:
      - name: telegram
        telegramConfigs:
          - botToken:
              name: telegram-bot-secret
              key: token
              chatID: -30490XXXXX
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: telegram
      repeatInterval: 12h

```

Поля `token` в Secret'е и `chatID` в ресурсе `CustomAlertmanager` необходимо поставить свои. Подробнее о Telegram API.

Пример отправки алертов в Slack с фильтром

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: slack
spec:
  internal:
    receivers:
      - name: devnull
      - name: slack
        slackConfigs:
          - apiURL:
              key: apiURL
              name: slack-apiurl
            channel: '{{ dig .Values.werf.env .Values.slack.channel._default .Values.slack.channel }}'
            fields:
              - short: true
                title: Severity
                value: '{{`{{ .CommonLabels.severity_level }}`}}'
              - short: true
                title: Status
                value: '{{`{{ .Status }}`}}'
              - title: Summary
                value: '{{`{{ range .Alerts }}`}}`{{`{{ .Annotations.summary }}`}}`{{`{{ end }}`}}`}}'
              - title: Description
                value: '{{`{{ range .Alerts }}`}}`{{`{{ .Annotations.description }}`}}`{{`{{ end }}`}}`}}'
              - title: Labels
                value: '{{`{{ range .Alerts }}`}}`{{`{{ range .Labels.SortedPairs }}`}}`{{`{{ printf "%s: %s\n" .Name .Value }}`}}`{{`{{ end }}`}}`{{`{{ end }}`}}`}}'
              - title: Links
                value: '{{`{{ (index .Alerts 0).GeneratorURL }}`}}'
                title: '{{`{{ .CommonLabels.alertname }}`}}'
    route:
      groupBy:
        - ''
      receiver: devnull
      routes:
        - matchers:
            - matchType: =~
              name: severity_level
              value: "^[4-9]$"
              receiver: slack
        repeatInterval: 12h
      type: Internal

```

Пример отправки алертов в Opsgenie

```
- name: opsgenie
  opsgenieConfigs:
  - apiKey:
      key: data
      name: opsgenie
    description: |
      {{ range .Alerts }}{{ .Annotations.summary }} {{ end }}
      {{ range .Alerts }}{{ .Annotations.description }} {{ end }}
    message: '{{ .CommonLabels.alertname }}'
    priority: P1
    responders:
      - id: team_id
        type: team
```

Пример отправки алертов по электронной почте

Создайте Secret с паролем от аккаунта электронной почты. Пароль, закодированный в формате base64, укажите в поле `password`:

```
apiVersion: v1
kind: Secret
metadata:
  name: am-mail-server-pass
  namespace: d8-monitoring
data:
  password: BASE64_ENCODED_PASSWORD_HERE
```

Измените значения в примере `CustomAlertManager` на необходимые вам и примените ресурс:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: mail
spec:
  type: Internal
  internal:
    receivers:
      - name: devnull
      - name: mail
    emailConfigs:
      - to: oncall@example.com
        from: prom@example.com
        smarthost: mx.example.com:587
        authIdentity: prom@example.com
        authUsername: prom@example.com
        authPassword:
          key: password
          name: am-mail-server-pass
        # Если вы используете custom CA на сервере, можете поместить публичную часть CA в ConfigMap в пространстве имен d8-monitoring
        # tlsConfig:
        #   insecureSkipVerify: true
        #   ca:
        #     configMap:
        #       key: ca.pem
        #       name: alertmanager-mail-server-ca
        sendResolved: true
        requireTLS: true
  route:
    groupBy:
      - job
    groupInterval: 5m
    groupWait: 30s
    receiver: devnull
    repeatInterval: 24h
    routes:
      - matchers:
          - matchType: =~
            name: severity_level
            value: "[1-4]"
        receiver: mail
```

FAQ

Как собирать метрики с приложений, расположенных вне кластера?

1. Сконфигурируйте Service по аналогии с сервисом для сбора метрик с вашего приложения, но без указания параметра `spec.selector`.
2. Создайте Endpoints для этого Service, явно указав в них `IP:PORT`, по которым ваши приложения отдают метрики.



Имена портов в Endpoints должны совпадать с именами этих портов в Service.

Пример

Метрики приложения доступны без TLS, по адресу `http://10.182.10.5:9114/metrics`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    prometheus.deckhouse.io/custom-target: my-app
spec:
  ports:
    - name: http-metrics
      port: 9114
---
apiVersion: v1
kind: Endpoints
metadata:
  name: my-app
  namespace: my-namespace
subsets:
  - addresses:
    - ip: 10.182.10.5
    ports:
    - name: http-metrics
      port: 9114
```

Как добавить дополнительные дашборды в вашем проекте?

Добавление пользовательских дашбордов для Grafana в Deckhouse реализовано с помощью подхода Infrastructure as a Code. Чтобы ваш дашборд появился в Grafana, необходимо создать в кластере специальный ресурс — `GrafanaDashboardDefinition`.

Пример:

```

apiVersion: deckhouse.io/v1
kind: GrafanaDashboardDefinition
metadata:
  name: my-dashboard
spec:
  folder: My folder # Папка, в которой в Grafana будет отображаться ваш дашборд.
  definition: |
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": "-- Grafana --",
            "enable": true,
            "hide": true,
            "iconColor": "rgba(0, 211, 255, 1)",
            "limit": 100,
            ...

```



Системные и добавленные через GrafanaDashboardDefinition дашборды нельзя изменить через интерфейс Grafana.

Алерты, настроенные в панели dashboard, не работают с шаблонами datasource — такой dashboard является невалидным и не импортируется. В версии Grafana 9.0 функционал legacy alerting был признан устаревшим и заменён на Grafana Alerting. В связи с этим, мы не рекомендуем использовать legacy alerting (оповещения панели мониторинга) в dashboards.



Если после применения дашборд не появляется в Grafana, возможно, в JSON файле дашборда присутствует ошибка. Чтобы определить источник проблемы, воспользуйтесь командой `d8 k logs -n d8-monitoring deployments/grafana-v10 dashboard-provisioner` для просмотра логов компонента, который осуществляет применение дашбордов.

Как добавить алерты и/или recording-правила для вашего проекта?

Для добавления алертов существует два способа:

- **Рекомендуемый способ** — с помощью ресурсов ClusterObservabilityMetricsRulesGroup и ObservabilityMetricsRulesGroup модуля observability.
- **Устаревший способ** — с помощью ресурса CustomPrometheusRules. Ниже приведён пример этого способа.

Параметры:

- `groups` — единственный параметр, в котором необходимо описать группы алертов. Структура групп полностью совпадает с аналогичной в prometheus-operator.

Пример:

```

apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  name: my-rules
spec:
  groups:
  - name: cluster-state-alert.rules
    rules:
    - alert: CephClusterErrorState
      annotations:
        description: Storage cluster is in error state for more than 10m.
        summary: Storage cluster is in error state
        plk_markup_format: markdown
      expr: |
        ceph_health_status{job="rook-ceph-mgr"} > 1

```

Как подключить дополнительные data source для Grafana?

Для подключения дополнительных data source к Grafana существует специальный ресурс — `GrafanaAdditionalDatasource`.

Параметры ресурса подробно описаны в документации к Grafana. Тип ресурса смотрите в документации по конкретному datasource.

Пример:

```

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: another-prometheus
spec:
  type: prometheus
  access: Proxy
  url: https://another-prometheus.example.com/prometheus
  basicAuth: true
  basicAuthUser: foo
  jsonData:
    timeInterval: 30s
    httpMethod: POST
  secureJsonData:
    basicAuthPassword: bar

```

Как обеспечить безопасный доступ к метрикам?

Для обеспечения безопасности настоятельно рекомендуем использовать `kube-rbac-proxy`.

Пример безопасного сбора метрик с приложения, расположенного в кластере

Чтобы настроить защиту метрик приложения с использованием `kube-rbac-proxy` и последующего сбора метрик с помощью Prometheus, выполните следующие шаги:

1. Создайте `ServiceAccount` с указанными ниже правами:

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rbac-proxy-test
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: rbac-proxy-test
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: d8:rbac-proxy
subjects:
- kind: ServiceAccount
  name: rbac-proxy-test
  namespace: default

```

i Обратите внимание, что используется встроенная в Deckhouse ClusterRole `d8:rbac-proxy`.

2. Создайте конфигурацию для `kube-rbac-proxy`:

```

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: rbac-proxy-config-test
  namespace: rbac-proxy-test
data:
  config-file.yaml: |+
    authorization:
      resourceAttributes:
        namespace: default
        apiVersion: v1
        resource: services
        subresource: proxy
        name: rbac-proxy-test

```

i Более подробную информацию по атрибутам можно найти в документации Kubernetes.

3. Создайте `Service` и `Deployment` для вашего приложения, где `kube-rbac-proxy` займет позицию `sidecar`-контейнера:

```

---
apiVersion: v1
kind: Service
metadata:
  name: rbac-proxy-test
  labels:
    prometheus.deckhouse.io/custom-target: rbac-proxy-test
spec:
  ports:
    - name: https-metrics
      port: 8443
      targetPort: https-metrics
  selector:
    app: rbac-proxy-test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rbac-proxy-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rbac-proxy-test
  template:
    metadata:
      labels:
        app: rbac-proxy-test
    spec:
      securityContext:
        runAsUser: 65532
      serviceAccountName: rbac-proxy-test
      containers:
        - name: kube-rbac-proxy
          image: quay.io/brancz/kube-rbac-proxy:v0.14.0
          args:
            - "--secure-listen-address=0.0.0.0:8443"
            - "--upstream=http://127.0.0.1:8081/"
            - "--config-file=/kube-rbac-proxy/config-file.yaml"
            - "--logtostderr=true"
            - "--v=10"
          ports:
            - containerPort: 8443
              name: https-metrics
          volumeMounts:
            - name: config
              mountPath: /kube-rbac-proxy
            - name: prometheus-example-app
              image: quay.io/brancz/prometheus-example-app:v0.1.0
              args:
                - "--bind=127.0.0.1:8081"
      volumes:
        - name: config
          configMap:
            name: rbac-proxy-config-test

```

4. Назначьте необходимые права на ресурс для Prometheus:

```

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rbac-proxy-test-client
rules:
- apiGroups: [""]
  resources: ["services/proxy"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: rbac-proxy-test-client
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rbac-proxy-test-client
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: d8-monitoring

```

После шага 4 метрики вашего приложения должны появиться в Prometheus.

Пример безопасного сбора метрик с приложения, расположенного вне кластера

Предположим, что есть доступный через интернет сервер, на котором работает `node-exporter`. По умолчанию `node-exporter` слушает на порту 9100 и доступен на всех интерфейсах. Необходимо обеспечить контроль доступа к `node-exporter` для безопасного сбора метрик. Ниже приведен пример такой настройки.

Требования:

- Из кластера должен быть доступ к сервису `kube-rbac-proxy`, который запущен на удаленном сервере.
- От удаленного сервера должен быть доступ к API-серверу кластера.

Выполните следующие шаги:

1. Создайте `ServiceAccount` с указанными ниже правами:


```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus-external-endpoint-server-01
  namespace: d8-service-accounts
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-external-endpoint
rules:
- apiGroups: ["authentication.k8s.io"]
  resources:
  - tokenreviews
  verbs: ["create"]
- apiGroups: ["authorization.k8s.io"]
  resources:
  - subjectaccessreviews
  verbs: ["create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus-external-endpoint-server-01
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus-external-endpoint
subjects:
- kind: ServiceAccount
  name: prometheus-external-endpoint-server-01
  namespace: d8-service-accounts

```

2. Сгенерируйте `kubeconfig` для созданного `ServiceAccount` (пример генерации `kubeconfig` для `ServiceAccount`).
3. Положите получившийся `kubeconfig` на *удаленный сервер*. Необходимо указать путь к этому `kubeconfig` в настройках `kube-rbac-proxy` (в примере используется путь `${PWD}/.kube/config`).
4. Настройте `node-exporter` на *удаленном сервере* с доступом к нему только на локальном интерфейсе (чтобы он слушал `127.0.0.1:9100`).
5. Запустите `kube-rbac-proxy` на *удаленном сервере*:

```

docker run --network host -d -v ${PWD}/.kube/config:/config quay.io/brancz/kube-rbac-proxy:v0.14.0 --secure-listen-address=0.0.0.0:8443 \
  --upstream=http://127.0.0.1:9100 --kubeconfig=/config --logtostderr=true --v=10

```

6. Проверьте, что порт `8443` доступен по внешнему адресу *удаленного сервера*.
7. Создайте в кластере `Service` и `Endpoint`, указав в качестве `<server_ip_address>` внешний адрес *удаленного сервера*:

```

---
apiVersion: v1
kind: Service
metadata:
  name: prometheus-external-endpoint-server-01
  labels:
    prometheus.deckhouse.io/custom-target: prometheus-external-endpoint-server-01
spec:
  ports:
    - name: https-metrics
      port: 8443
---
apiVersion: v1
kind: Endpoints
metadata:
  name: prometheus-external-endpoint-server-01
subsets:
  - addresses:
    - ip: <server_ip_address>
    ports:
    - name: https-metrics
      port: 8443

```

Как добавить Alertmanager?

Создайте кастомный ресурс `CustomAlertmanager` с типом `Internal`.

Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: webhook
spec:
  type: Internal
  internal:
    route:
      groupBy: ['job']
      groupWait: 30s
      groupInterval: 5m
      repeatInterval: 12h
      receiver: 'webhook'
    receivers:
      - name: 'webhook'
        webhookConfigs:
          - url: 'http://webhookserver:8080/'

```

Подробнее о всех параметрах можно прочитать в описании кастомного ресурса `CustomAlertmanager`.

Как добавить внешний дополнительный Alertmanager?

Создайте кастомный ресурс `CustomAlertmanager` с типом `External`, который может указывать на Alertmanager по FQDN или через сервис в Kubernetes-кластере.

Пример FQDN Alertmanager:

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: my-fqdn-alertmanager
spec:
  external:
    address: https://alertmanager.mycompany.com/myprefix
    type: External

```

Пример Alertmanager с Kubernetes service:

```

apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: my-service-alertmanager
spec:
  external:
    service:
      namespace: myns
      name: my-alertmanager
      path: /myprefix/
    type: External

```

Подробно о всех параметрах можно прочитать в описании кастомного ресурса CustomAlertmanager.

Как в Alertmanager игнорировать лишние алерты?

Решение сводится к настройке маршрутизации алертов в вашем Alertmanager.

Требования:

1. Завести получателя без параметров.
2. Смаршрутизировать лишние алерты в этого получателя.

Ниже приведены примеры настройки CustomAlertmanager.

Чтобы получать только алерты с лейблами `service: foo|bar|baz`:

```

receivers:
  # Получатель, определенный без параметров, будет работать как "/dev/null".
  - name: blackhole
  # Действующий получатель
  - name: some-other-receiver
  # ...
route:
  # receiver по умолчанию.
  receiver: blackhole
routes:
  # Дочерний маршрут
  - matchers:
    - matchType: =~
      name: service
      value: ^(foo|bar|baz)$
    receiver: some-other-receiver

```

Чтобы получать все алерты, кроме `DeadMansSwitch`:

```

receivers:
  # Получатель, определенный без параметров, будет работать как "/dev/null".
  - name: blackhole
  # Действующий получатель.
  - name: some-other-receiver
  # ...
route:
  # receiver по умолчанию.
  receiver: some-other-receiver
routes:
  # Дочерний маршрут.
  - matchers:
    - matchType: =
      name: alertname
      value: DeadMansSwitch
    receiver: blackhole

```

С подробным описанием всех параметров можно ознакомиться в официальной документации.

Почему нельзя установить разный scrapeInterval для отдельных таргетов?

Наиболее полный ответ на этот вопрос дает разработчик Prometheus Brian Brazil. Вкратце, разные scrapeInterval'ы принесут следующие проблемы:

- увеличение сложности конфигурации;
- проблемы при написании запросов и создании графиков;
- короткие интервалы больше похожи на профилирование приложения, и, скорее всего, Prometheus — не самый подходящий инструмент для этого.

Наиболее подходящее значение для scrapeInterval находится в диапазоне 10–60 секунд.

Как ограничить потребление ресурсов Prometheus?

Чтобы предотвратить ситуации, когда Variable Policy Agent (VPA) запрашивает у Prometheus или долгосрочного Prometheus больше ресурсов, чем доступно на выделенном узле для этих целей, можно явно установить ограничения для VPA с использованием параметров модуля:

- `vpa.longtermMaxCPU`;
- `vpa.longtermMaxMemory`;
- `vpa.maxCPU`;
- `vpa.maxMemory`.

Как настроить ServiceMonitor или PodMonitor для работы с Prometheus?

Добавьте лейбл `prometheus: main` к Pod/Service Monitor. Добавьте в пространство имён, в котором находится Pod/Service Monitor, лейбл `prometheus.deckhouse.io/monitor-watcher-enabled: "true"`.

Пример:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/monitor-watcher-enabled: "true"
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      app: example-app
  endpoints:
    - port: web

```

Как настроить Probe для работы с Prometheus?

Добавьте лейбл `prometheus: main` к Probe. Добавьте в пространство имён, в котором находится Probe, лейбл `prometheus.deckhouse.io/probe-watcher-enabled: "true"`.

Пример:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/probe-watcher-enabled: "true"
---
apiVersion: monitoring.coreos.com/v1
kind: Probe
metadata:
  labels:
    app: prometheus
    component: probes
    prometheus: main
  name: cdn-is-up
  namespace: frontend
spec:
  interval: 30s
  jobName: httpGet
  module: http_2xx
  prober:
    path: /probe
    scheme: http
    url: blackbox-exporter.blackbox-exporter.svc.cluster.local:9115
  targets:
    staticConfig:
      static:
        - https://example.com/status

```

Как настроить PrometheusRules для работы с Prometheus?

Добавьте в пространство имён, в котором находятся PrometheusRules, лейбл `prometheus.deckhouse.io/rules-watcher-enabled: "true"`.

Пример:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/rules-watcher-enabled: "true"
```

Как увеличить размер диска

1. Для увеличения размера отредактируйте PersistentVolumeClaim, указав новый размер в поле `spec.resources.requests.storage`.
 - Увеличение размера возможно, если в StorageClass поле `allowVolumeExpansion` установлено в `true`.
2. Если используемое хранилище не поддерживает изменение диска на лету, в статусе PersistentVolumeClaim появится сообщение `Waiting for user to (re-)start a pod to finish file system resize of volume on node..`
3. Перезапустите под для завершения изменения размера файловой системы.

Как получить информацию об алертах в кластере?

Информацию об активных алертах можно получить не только в веб-интерфейсе Grafana/Prometheus, но и в CLI. Это может быть полезным, если у вас есть только доступ к API-серверу кластера и нет возможности открыть веб-интерфейс Grafana/Prometheus.

Выполните следующую команду для получения списка алертов в кластере:

```
d8 k get clusteralerts
```

Пример вывода:

NAME	ALERT	SEVERITY	AGE
LAST RECEIVED	STATUS		
086551ae5b5b24	ExtendedMonitoringDeprecatatedAnnotation	4	3h25m 38s
firing			
226d35c886464d6e	ExtendedMonitoringDeprecatatedAnnotation	4	3h25m 38s
firing			
235d4efba7df6af4	D8SnapshotControllerPodIsNotReady	8	5d4h 44s
firing			
27464763f0aa857c	D8PrometheusOperatorPodIsNotReady	7	5d4h 43s
firing			
ab17837fffa5e440	DeadMansSwitch	4	5d4h 41s
firing			

Выполните следующую команду для просмотра конкретного алерта:

```
d8 k get clusteralerts <ALERT_NAME> -O yaml
```

Пример:

```
# d8 k get clusteralerts 235d4efba7df6af4 -o yaml
alert:
  description: |
    The recommended course of action:
    1. Retrieve details of the Deployment: `kubectl -n d8-snapshot-controller describe deploy snapshot-controller`
    2. View the status of the Pod and try to figure out why it is not running: `kubectl -n d8-snapshot-controller describe pod -l app=snapshot-controller`
  labels:
    pod: snapshot-controller-75bd776d76-xhb2c
    prometheus: deckhouse
    tier: cluster
  name: D8SnapshotControllerPodIsNotReady
  severityLevel: "8"
  summary: The snapshot-controller Pod is NOT Ready.
  apiVersion: deckhouse.io/v1alpha1
  kind: ClusterAlert
  metadata:
    creationTimestamp: "2023-05-15T14:24:08Z"
    generation: 1
    labels:
      app: prometheus
      heritage: deckhouse
      name: 235d4efba7df6af4
      resourceVersion: "36262598"
      uid: 817f83e4-d01a-4572-8659-0c0a7b6ca9e7
  status:
    alertStatus: firing
    lastUpdateTime: "2023-05-15T18:10:09Z"
    startsAt: "2023-05-10T13:43:09Z"
```



Присутствие специального алерта `MissingDeadMansSwitch` в кластере говорит о проблемах в работоспособности компонентов мониторинга.

Список всех доступных алертов системы мониторинга Deckhouse Kubernetes Platform приведён на отдельной странице документации.

Как добавить дополнительные эндпоинты в scrape config?

Добавьте в пространство имён, в котором находится `ScrapeConfig`, лейбл `prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"`.

Пример:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"
```

Добавьте `ScrapeConfig`, который имеет обязательный лейбл `prometheus: main`:

```

apiVersion: monitoring.coreos.com/v1alpha1
kind: ScrapeConfig
metadata:
  name: example-scrape-config
  namespace: frontend
  labels:
    prometheus: main
spec:
  honorLabels: true
  staticConfigs:
    - targets: ['example-app.frontend.svc.{{ .Values.global.discovery.clusterDomain }}.:8080']
  relabelings:
    - regex: endpoint|namespace|pod|service
      action: labeldrop
    - targetLabel: scrape_endpoint
      replacement: main
    - targetLabel: job
      replacement: kube-state-metrics
  metricsPath: '/metrics'

```

Модуль prometheus-metrics-adapter

Позволяет работать HPA- и VPA-автоскейлерам по «любым» метрикам.

Устанавливает в кластер имплементацию Kubernetes resource metrics API, custom metrics API и external metrics API для получения метрик из Prometheus.

Это позволяет:

- `kubectl top` брать метрики из Prometheus, через адаптер;
- использовать custom resource версии autoscaling/v2 для масштабирования приложений (HPA);
- получать информацию из Prometheus средствами API Kubernetes для других модулей (Vertical Pod Autoscaler, ...).

Модуль позволяет производить масштабирование по следующим параметрам:

- CPU (пода);
- память (пода);
- rps (Ingress'a) — за 1, 5, 15 минут (`rps_Nm`);
- CPU (пода) — за 1, 5, 15 минут (`cpu_Nm`) — среднее потребление CPU за N минут;
- память (пода) — за 1, 5, 15 минут (`memory_Nm`) — среднее потребление памяти за N минут;
- любые Prometheus-метрики и любые запросы на их основе.

Как работает

Данный модуль регистрирует `k8s-prometheus-adapter` в качестве external API-сервиса, который расширяет возможности Kubernetes API. Когда какому-то из компонентов Kubernetes (VPA, HPA) требуется информация об используемых ресурсах, он делает запрос в Kubernetes API, а тот, в свою очередь, проксирует запрос в адаптер. Адаптер на основе своего конфигурационного файла выясняет, как посчитать метрику, и отправляет запрос в Prometheus.

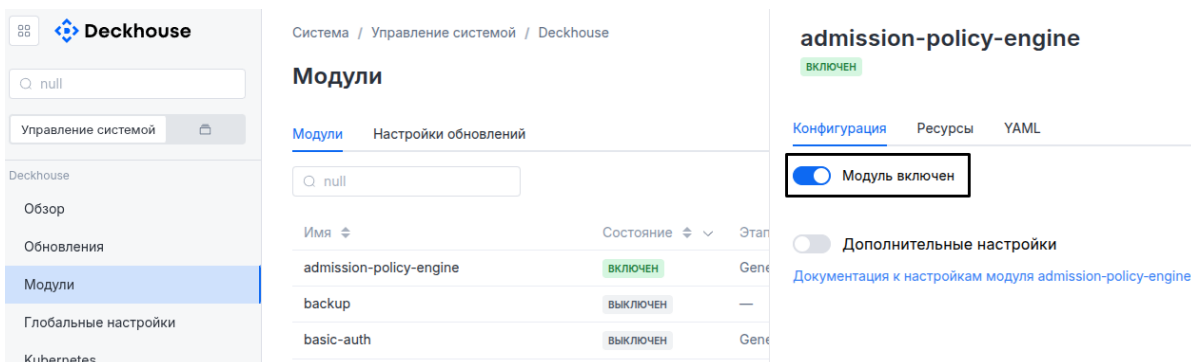
Настройки

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-metrics-adapter`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `prometheus-metrics-adapter`:

```
d8 system module enable prometheus-metrics-adapter
```

- **С помощью ModuleConfig prometheus-metrics-adapter.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `prometheus-metrics-adapter` (создайте его, при необходимости).

Пример манифеста для включения модуля `prometheus-metrics-adapter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-metrics-adapter
spec:
  enabled: true
```

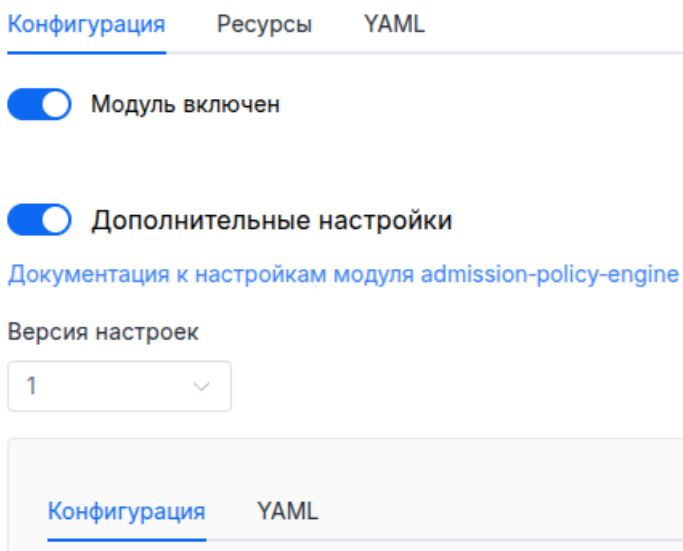
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-metrics-adapter`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `prometheus-metrics-adapter` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-metrics-adapter`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `prometheus-metrics-adapter` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc prometheus-metrics-adapter
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `prometheus-metrics-adapter`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `prometheus-metrics-adapter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-metrics-adapter
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

В общем случае конфигурации не требуется.

Параметры

- `highAvailability` — ручное включение/отключение режима отказоустойчивости. По умолчанию режим отказоустойчивости определяется автоматически. Смотри подробнее про режим отказоустойчивости для модулей.

Custom resources

i

Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Настройка ванильного `prometheus-metrics-adapter` — трудоемкий процесс. Мы его упростили, определив набор **CustomResourceDefinition** с разной областью видимости (scope).

С помощью cluster-wide-ресурса можно определить метрику глобально, а с помощью namespaced-ресурса ее можно локально переопределять. Формат всех кастомных ресурсов — одинаковый.

Namespaced custom resources

ServiceMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", service="myservice"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, service` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

IngressMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", ingress="myingress"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, ingress` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

PodMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", pod="mypod-xxxxx"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, pod` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

DeploymentMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", deployment="mydeployment"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, deployment` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

StatefulSetMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", statefulset="mystatefulset"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, statefulset` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

NamespaceMetric

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

DaemonSetMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", daemonset="mydaemonset"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, daemonset` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

Cluster custom resources**ClusterServiceMetric (недоступен пользователям)**

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", service="myservice"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, service` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

ClusterIngressMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", ingress="myingress"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, ingress` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

ClusterPodMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", pod="mypod-xxxxx"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, pod` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

ClusterDeploymentMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", deployment="mydeployment"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, deployment` для группировки (`max()` `by(...)`, `sum()` `by (...)` и `pr()`).

ClusterStatefulSetMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и `pr()`). В запросе необходимо **обязательно использовать** ключи:

- `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", statefulset="mystatefulset"}`. Можно добавить собственные лейблы через запятую (пример).
- `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, statefulset` для группировки (`max()` `by(...)`, `sum()` `by (...)` и пр.).

ClusterDaemonSetMetric (недоступен пользователям)

- `.metadata.name` — имя метрики, используется в HPA.
- `.spec.query` — кастомный PromQL-запрос, который возвращает однозначное значение для вашего набора лейблов (используйте группировку операторами `sum()` `by()`, `max()` `by()` и пр.). В запросе необходимо **обязательно использовать** ключи:
 - `<<.LabelMatchers>>` — заменится на набор лейблов `{namespace="mynamespace", daemonset="mydaemonset"}`. Можно добавить собственные лейблы через запятую (пример).
 - `<<.GroupBy>>` — заменится на перечисление лейблов `namespace, daemonset` для группировки (`max()` `by(...)`, `sum()` `by (...)` и пр.).

Примеры конфигурации

Ниже рассматривается только HPA (Horizontal Pod Autoscaling) с `apiVersion: autoscaling/v2`, чья поддержка появилась начиная с Kubernetes v1.12.

Для настройки HPA требуется:

- определить, что именно масштабируется (`.spec.scaleTargetRef`);
- определить диапазон масштабирования (`.spec.minReplicas`, `.scale.maxReplicas`);
- зарегистрировать в API Kubernetes и определить метрики, на основе которых производится масштабирование (`.spec.metrics`).

Метрики с точки зрения HPA бывают трех видов:

- классические — с типом (`.spec.metrics[].type`) «Resource», используются для простейшего масштабирования по потреблению процессора и памяти;
- кастомные — с типами (`.spec.metrics[].type`) «Pods» или «Object»;
- внешние — с типом (`.spec.metrics[].type`) «External».

Важно! По умолчанию HPA использует разные подходы при масштабировании:

- Если метрики указывают на требование масштабировать **вверх**, это происходит незамедлительно (`spec.behavior.scaleUp.stabilizationWindowSeconds = 0`). Единственное ограничение — скорость прироста: за 15 секунд поды могут удвоиться, но если подов меньше 4, добавятся 4 новых пода.
- Если метрики указывают на то, что требуется масштабировать **вниз**, это происходит в течение 5 минут (`spec.behavior.scaleUp.stabilizationWindowSeconds = 300`): собираются предложения о новом количестве реплик, в результате чего выбирается самое большое значение. Нет ограничений на количество удаляемых подов за один раз.

Если имеются проблемы с колебаниями метрик и происходит резкое увеличение ненужных реплик приложения, применяются следующие подходы:

- Оборачивание метрики агрегирующей функцией (например, `avg_over_time()`), если метрика определена PromQL-запросом. Подробнее см. пример.
- Увеличение времени стабилизации (параметр `spec.behavior.scaleUp.stabilizationWindowSeconds`) в ресурсе *HorizontalPodAutoscaler*. В течение обозначенного периода будут собираться предложения об увеличении количества реплик, в результате чего будет выбрано самое скромное предложение. Это решение тождественно применению агрегирующей функции `min_over_time(<stabilizationWindowSeconds>)`, но только в том случае, если метрика растёт и требуется масштабирование **вверх**. Для масштабирования **вниз**, как правило, достаточно стандартных настроек. Подробнее см. пример.
- Ограничение скорости прироста новых реплик с помощью политик `spec.behavior.scaleUp.policies`.

Типы масштабирования

Используйте следующие метрики для масштабирования приложений:

1. Классического типа.
2. Кастомные `namespace-scored`-метрики. При условии, если у вас одно приложение, источник метрик находится внутри `namespace` и связан с одним из объектов.
3. Кастомные `cluster-wide`-метрики. При условии, если у вас много приложений используют одинаковую метрику, источник которой находится в `namespace` приложения, и она связана

- с одним из объектов. Подобные метрики предусмотрены на случай необходимости выделения общих инфраструктурных компонентов в отдельный деплой («infra»).
4. Если источник метрики не привязан к namespace приложения, используйте внешние метрики. Например, метрики облачного провайдера или внешнего SaaS-сервиса.

Важно! Рекомендуется использовать вариант 1 (классические метрики), или вариант 2 (кастомные метрики, определяемые в *Namespace*). В этом случае, рекомендуется определить конфигурацию приложения, включающую его автоматическое масштабирование, в репозиторий самого приложения. Следует рассматривать варианты 3 и 4 только в том случае, если у вас имеется большая коллекция идентичных микросервисов.

Классическое масштабирование по потреблению ресурсов

Пример HPA для масштабирования по базовым метрикам из `metrics.k8s.io`: CPU и памяти подов. Особое внимание на `averageUtilization` — это значение отражает целевой процент ресурсов, который был **реквестирован**.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: app-prod
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  # Границы масштабирования контроллера.
  minReplicas: 1
  maxReplicas: 10
  # Если для приложения характерны кратковременные скачки потребления CPU,
  # можно отложить принятие решения о масштабировании, чтобы убедиться, что оно
  необходимо.
  # По умолчанию масштабирование вверх происходит немедленно.
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 300
  metrics:
    # Масштабирование по CPU и памяти.
    - type: Resource
      resource:
        name: cpu
        target:
          # Масштабирование, когда среднее использование CPU всех подов в scaleTargetRef
          превышает заданное значение.
          # Для метрики с type: Resource доступен только type: Utilization.
          type: Utilization
          # Масштабирование, если для всех подов из Deployment запрошено по 1 ядру и в
          среднем уже используется более 700m.
          averageUtilization: 70
    - type: Resource
      resource:
        name: memory
        target:
          # Пример масштабирования, когда среднее использование памяти всех подов в
          scaleTargetRef превышает заданное значение.
          type: Utilization
          # Масштабирование, если для подов запрошено по 1 ГБ памяти и в среднем
          использовано уже более 800 МБ.
          averageUtilization: 80
```

Масштабирование по кастомным метрикам

Регистрация кастомных метрик в Kubernetes API

Кастомные метрики необходимо регистрировать в API `/apis/custom.metrics.k8s.io/`, эту регистрацию производит `prometheus-metrics-adapter` (и он же реализует API). На эти метрики можно будет ссылаться из объекта *HorizontalPodAutoscaler*. Настройка ванильного `prometheus-metrics-adapter` — трудоемкий процесс, мы его упростили, определив набор Custom Resources с разным Scope:

- Namespaced:
 - `ServiceMetric`;
 - `IngressMetric`;
 - `PodMetric`;
 - `DeploymentMetric`;
 - `StatefulsetMetric`;
 - `NamespaceMetric`;
 - `DaemonSetMetric` (недоступен пользователям).
- Cluster:
 - `ClusterServiceMetric` (недоступен пользователям);
 - `ClusterIngressMetric` (недоступен пользователям);
 - `ClusterPodMetric` (недоступен пользователям);
 - `ClusterDeploymentMetric` (недоступен пользователям);
 - `ClusterStatefulsetMetric` (недоступен пользователям);
 - `ClusterDaemonSetMetric` (недоступен пользователям).

С помощью cluster-wide-ресурса можно задать глобальное определение метрики, а с помощью *Namespace* можно переопределить её локально. Формат для всех custom resource — одинаковый.

Применяем кастомные метрики в НРА

После регистрации кастомной метрики на нее можно ссылаться. С точки зрения НРА, кастомные метрики бывают двух видов — `Pods` и `Object`.

`Object` — отсылает к объекту в кластере, который имеет в Prometheus метрики с соответствующими лейблами (`namespace=XXX,ingress=YYY`). Эти лейблы будут подставляться вместо `<<.LabelMatchers>>` в вашем кастомном запросе.

```

apiVersion: deckhouse.io/v1beta1
kind: IngressMetric
metadata:
  name: mymetric
  namespace: mynamespace
spec:
  query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by
(<<.GroupBy>>) OR on() vector(0)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  # Метрики, используемые для масштабирования.
  # Пример использования кастомных метрик.
  metrics:
  - type: Object
    object:
      # Объект, который обладает метриками в Prometheus.
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: myingress
      metric:
        # Метрика, зарегистрированная с помощью custom resource IngressMetric или
ClusterIngressMetric.
        # Можно использовать rps_1m, rps_5m или rps_15m которые поставляются с модулем
prometheus-metrics-adapter.
        name: mymetric
      target:
        # Для метрик типа Object можно использовать `Value` или `AverageValue`.
        type: AverageValue
        # Масштабирование происходит, если среднее значение кастомной метрики для всех
подов в Deployment сильно отличается от 10.
        averageValue: 10

```

Pods — из ресурса, которым управляет НРА, будут выбраны все поды и для каждого пода будут собраны метрики с соответствующими лейблами (`namespace=XXX`, `pod=YYY-sadiq`, `namespace=XXX`, `pod=YYY-e3adf`, и т. д.). Из этих показателей НРА рассчитает среднее значение и использует для масштабирования.

Пример использования кастомных метрик с размером очереди RabbitMQ

В представленном примере рассматривается очередь `send_forum_message` в RabbitMQ, для которого зарегистрирован сервис `rmq`. Если количество сообщений в этой очереди превышает 42, выполняется масштабирование.


```

apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"})
by (<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42

```

Пример использования нестабильной кастомной метрики

Улучшение предыдущего примера.

В представленном примере рассматривается очередь `send_forum_message` в RabbitMQ, для которого зарегистрирован сервис `rmq`. Если количество сообщений в этой очереди превышает 42, выполняется масштабирование. Мы не хотим реагировать на краткосрочные всплески, поэтому используется MQL-функцию `avg_over_time()`, чтобы усреднить метрику.

```

apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
  (avg_over_time(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"}[5m])) by (<<.GroupBy>>)
  ---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42

```

Примеры с использованием кастомных метрик типа `Pods`

Пример масштабирования воркеров по процентному количеству активных php-fpm-воркеров. В представленном примере среднее количество php-fpm-воркеров в *Deployment* `mybackend` не больше 5.

```

apiVersion: deckhouse.io/v1beta1
kind: PodMetric
metadata:
  name: php-fpm-active-workers
spec:
  query: sum (phpfpm_processes_total{state="active",<<.LabelMatchers>>}) by
(<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mybackend
  minReplicas: 1
  maxReplicas: 5
  metrics:
    # Указание HPA обойти все поды Deployment'a и собрать с них метрики.
    - type: Pods
      # Указывать describedObject в отличие от type: Object не надо.
      pods:
        metric:
          # Кастомная метрика, зарегистрированная с помощью custom resource PodMetric.
          name: php-fpm-active-workers
        target:
          # Для метрик с type: Pods можно использовать только AverageValue.
          type: AverageValue
          # Масштабирование, если среднее значение метрики у всех подов Deployment'a
          больше 5.
          averageValue: 5

```

Масштабируется Deployment по процентному количеству активных php-fpm-воркеров.

```

---
apiVersion: deckhouse.io/v1beta1
kind: PodMetric
metadata:
  name: php-fpm-active-worker
spec:
  # Процент активных php-fpm-воркеров. Функция round() для того, чтобы не смущаться
  # от миллипроцентов в HPA.
  query: round(sum by(<<.GroupBy>>) (phpfpm_processes_total{state="active",
  <<.LabelMatchers>>}) / sum by(<<.GroupBy>>)
  (phpfpm_processes_total{<<.LabelMatchers>>}) * 100)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: {{ .Chart.Name }}-hpa
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  # statefulset).
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: {{ .Chart.Name }}
  minReplicas: 4
  maxReplicas: 8
  metrics:
  - type: Pods
    pods:
      metric:
        name: php-fpm-active-worker
      target:
        type: AverageValue
        # Масштабирование, если в среднем по Deployment 80% воркеров заняты.
        averageValue: 80

```

Регистрация внешних метрик в Kubernetes API

Модуль `prometheus-metrics-adapter` поддерживает механизм `externalRules`, с помощью которого можно определять кастомные PromQL-запросы и регистрировать их как метрики.

В примерах инсталляций добавлено универсальное правило, которое позволяет создавать собственные метрики без настроек в `prometheus-metrics-adapter`, — «любая метрика в Prometheus с именем `kube_adapter_metric_<name>` будет зарегистрирована в API под именем `<name>`». После чего, остается написать экспортер (`exporter`), который будет экспортировать подобную метрику, или создать правило `recording rule` в Prometheus, которое будет агрегировать вашу метрику на основе других метрик.

Пример *CustomPrometheusRules*:

В примере представлены пользовательские правила Prometheus для метрики `mymetric`.

```

apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  # Рекомендованный шаблон для названия ваших CustomPrometheusRules.
  name: prometheus-metrics-adapter-mymetric
spec:
  groups:
    # Рекомендованный шаблон.
    - name: prometheus-metrics-adapter.mymetric
      rules:
        # Название вашей новой метрики.
        # Важно! Префикс 'kube_adapter_metric_' обязателен.
        - record: kube_adapter_metric_mymetric
          # Запрос, результаты которого попадут в итоговую метрику, нет смысла тащить в
          # нее лишние лейблы.
          expr: sum(ingress_nginx_detail_sent_bytes_sum) by (namespace,ingress)

```

Применение внешних метрик в HPA

После регистрации внешней метрики на нее можно сослаться.

```

kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  # statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  metrics:
    # Используем внешние метрики для масштабирования.
    - type: External
      external:
        metric:
          # Метрика, которую мы зарегистрировали с помощью создания метрики в Prometheus
          # kube_adapter_metric_mymetric, но без префикса 'kube_adapter_metric_'.
          name: mymetric
          selector:
            # Для внешних метрик можно и нужно уточнять запрос с помощью лейблов.
            matchLabels:
              namespace: mynamespace
              ingress: myingress
        target:
          # Для метрик типа External можно использовать только `type: Value`.
          type: Value
          # Масштабирование, если значение нашей метрики больше 10.
          value: 10

```

Пример с размером очереди в Amazon SQS

Чтобы установить экспортер для интеграции с SQS:

1. Создайте отдельный «служебный» репозиторий Git (или, к примеру, можно использовать «инфраструктурный» репозиторий).
2. Разместите в нем установку экспортера и сценарий для создания требуемого

CustomPrometheusRules.

Готово, вы объединили кластер. Если необходимо настроить автомасштабирование только для одного приложения (в одном неймспейсе), лучше ставить экспортер вместе с этим приложением и воспользоваться `NamespaceMetrics`.

Ниже приведен пример экспортера (например, `sqs-exporter`) для получения метрик из Amazon SQS, если:

- в Amazon SQS работает очередь `send_forum_message`;
- выполняется масштабирование при количестве сообщений в этой очереди больше 42.

```
apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  # Рекомендованное название – prometheus-metrics-adapter-<metric name>.
  name: prometheus-metrics-adapter-sqs-messages-visible
spec:
  groups:
    # Рекомендованный шаблон названия.
    - name: prometheus-metrics-adapter.sqs_messages_visible
      rules:
        # Важно! Префикс 'kube_adapter_metric_' обязателен.
        - record: kube_adapter_metric_sqs_messages_visible
          expr: sum (sqs_messages_visible) by (queue)
  ---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Указывается контроллер, который нужно масштабировать (ссылка на deployment или
  statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: External
      external:
        metric:
          # name должен совпадать с CustomPrometheusRules record без префикса
          'kube_adapter_metric_'.
          name: sqs_messages_visible
          selector:
            matchLabels:
              queue: send_forum_messages
        target:
          type: Value
          value: 42
```

Способы отладки**Как получить список кастомных метрик?**

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/
```

Как получить значение метрики, привязанной к объекту?

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/services/*/my-service-metric
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/ingresses/*/rps_1m
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/ingresses/*/mymetric
```

Как получить значение метрики, созданной через NamespaceMetric?

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/metrics/my-ns-metric
```

Как получить external-метрики?

```
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1/namespaces/d8-ingress-nginx/d8_ingress_nginx_ds_cpu_utilization
```

Модуль Prometheus Pushgateway

Этот модуль устанавливает в кластер Prometheus Pushgateway. Он предназначен для приема метрик от приложения и отдачи их Prometheus.

Дополнительная информация:

- Когда необходимо использовать Prometheus Pushgateway.
- Как собирать метрики с помощью Prometheus Pushgateway.

Настройки

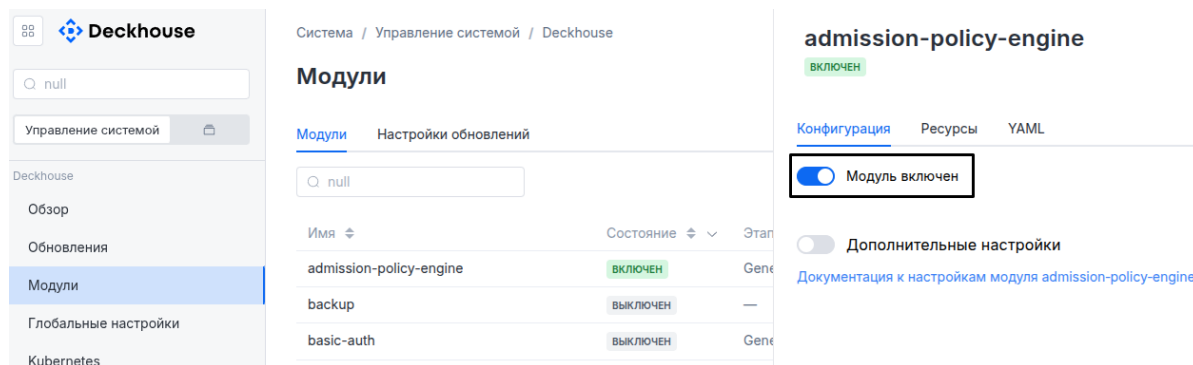
Данный модуль устанавливает в кластер Prometheus Pushgateway. Он предназначен для приема метрик от приложения и отдачи их Prometheus.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-pushgateway`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `prometheus-pushgateway`:

```
d8 system module enable prometheus-pushgateway
```

- **С помощью ModuleConfig prometheus-pushgateway.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `prometheus-pushgateway` (создайте его, при необходимости).

Пример манифеста для включения модуля `prometheus-pushgateway`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  enabled: true
```

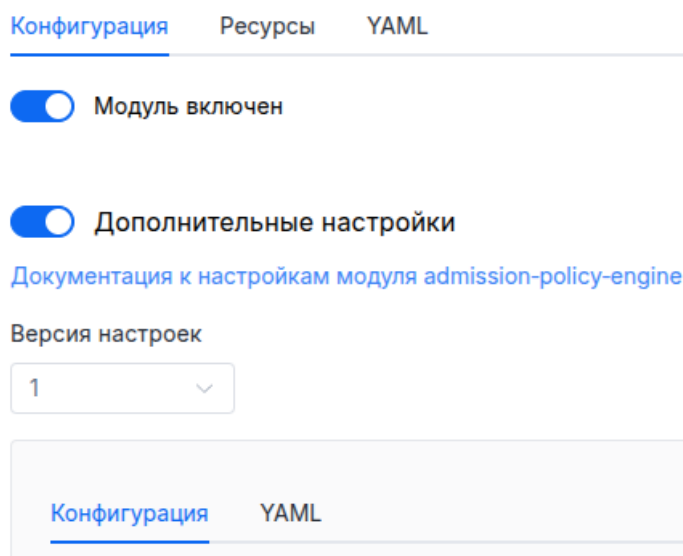
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-pushgateway`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект `ModuleConfig prometheus-pushgateway` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `prometheus-pushgateway`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig prometheus-pushgateway` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc prometheus-pushgateway
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig prometheus-pushgateway`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig prometheus-pushgateway`:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Требования

К версиям других модулей:

- `prometheus`: любая версия.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.instances**

массив строк

Обязательный параметр

Содержит список инстансов, для каждого из которых будет создан отдельный PushGateway.

Пример:

```

instances:
- first
- second

```

- **Элемент массива**

строка

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

Максимальная длина: 63

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**

строка

- **settings.tolerations.operator**

строка

- **settings.tolerations.tolerationSeconds**

целочисленный

- **settings.tolerations.value**

строка

Примеры

Пример настройки модуля

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  version: 1
  enabled: true
  settings:
    instances:
      - first
      - second
      - another

```

Адрес PushGateway (из контейнера пода): `http://first.kube-prometheus-pushgateway:9091`.

Отправка метрики

Пример отправки метрики через curl:

```

echo "test_metric{env="dev"} 3.14" | curl --data-binary @- http://first.kube-prometheus-
pushgateway:9091/metrics/job/myapp

```

Через 30 секунд (после скрейпа данных) метрики будут доступны в Prometheus. Пример:

```

test_metric{container="prometheus-pushgateway", env="dev", exported_job="myapp",
  instance="10.244.1.155:9091", job="prometheus-pushgateway", pushgateway="prometheus-pushgateway",
  tier="cluster"} 3.14

```

i Название job (в примере — `myapp`) будет доступно в Prometheus в лейбле `exported_job`, а не `job` (так как лейбл `job` уже занят в Prometheus, он переименовывается при приеме метрики от PushGateway).

i Возможно, вам потребуется получить список всех имеющихся job для выбора уникального названия (чтобы не испортить существующие графики и алерты). Получить список всех имеющихся job можно следующим запросом: `count({__name__=~".+"}) by (job)`.

Удаление метрик

Пример удаления всех метрик группы `{instance="10.244.1.155:9091", job="myapp"}` через curl:

```

curl -X DELETE http://first.kube-prometheus-
pushgateway:9091/metrics/job/myapp/instance/10.244.1.155:9091

```

Так как PushGateway хранит полученные метрики в памяти, **при рестарте пода все метрики будут утеряны**.

Модуль registry



Стадия жизненного цикла модуля: Preview

Описание

Модуль отвечает за управление настройками registry компонентов DKP.

Модуль работает в следующих режимах:

- **Direct** — использование прямого доступа к внешнему registry по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse`. Фиксированный адрес, при изменении параметров registry, позволяет избежать повторного скачивания образов и перезапуска компонентов при смене параметров registry. Переключение между режимами и registry

выполняется через `ModuleConfig deckhouse`. Переключение выполняется автоматически (ознакомьтесь с примерами использования).

- **Proxy** — использование внутреннего кеширующего прокси-registry с обращением к внешнему registry, с запуском кеширующего прокси-registry на control-plane (master) узлах. Режим позволяет сократить количество запросов к внешнему registry за счёт кеширования образов. Кешируемые данные хранятся на control-plane (master) узлах. Обращение к внутреннему registry выполняется по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse` аналогично Direct режиму. Переключение между режимами и registry выполняется через `ModuleConfig deckhouse`. Переключение выполняется автоматически (ознакомьтесь с примерами использования).
- **Local** — использование локального внутреннего registry, с запуском registry на control-plane (master) узлах. Режим позволяет кластеру работать в изолированной среде. Данные хранятся на control-plane (master) узлах. Обращение к внутреннему registry выполняется по фиксированному адресу `registry.d8-system.svc:5001/system/deckhouse` аналогично Direct и Proxy режимам. Переключение между режимами и registry выполняется через `ModuleConfig deckhouse`. Переключение выполняется автоматически (ознакомьтесь с примерами использования).
- **Unmanaged** — работа без использования внутреннего registry. Обращение внутри кластера выполняется напрямую к внешнему registry. Существует 2 вида режима **Unmanaged**:
 - Конфигурируемый - режим, управляемый с помощью модуля `registry`. Переключение между режимами и registry выполняется через `ModuleConfig deckhouse`. Переключение выполняется автоматически (ознакомьтесь с примерами использования).
 - Неконфигурируемый (deprecated) - режим используемый по умолчанию. Параметры конфигурации задаются при установке кластера, или при изменении в развёрнутом кластере с помощью утилиты `helper change registry (deprecated)`.

Ограничения и особенности использования модуля

Модуль `registry` имеет ряд ограничений и особенностей, касающихся установки, условий работы и переключения режимов.

Ограничения при установке кластера

- Bootstrap кластера DKP поддерживается только в **Direct**, **Unmanaged** и **Proxy** режимах (bootstrap в **Local** режиме не поддерживается). Registry во время установки кластера настраивается через `ModuleConfig deckhouse`.
- Для запуска кластера в неконфигурируемом **Unmanaged** режиме (Legacy), необходимо указать параметры registry в `initConfiguration`.

Ограничения по условиям работы

Модуль работает при соблюдении следующих условий:

- Если на узлах кластера используется CRI containerd или containerd v2. Для настройки CRI ознакомьтесь с конфигурацией `ClusterConfiguration`.
- Кластер полностью управляется DKP. В Managed Kubernetes кластерах он работать не будет.
- Работа режимов **Local** и **Proxy** поддерживаются только на статичных кластерах.

Ограничения по переключению режимов

Ограничения по переключению режимов следующие:

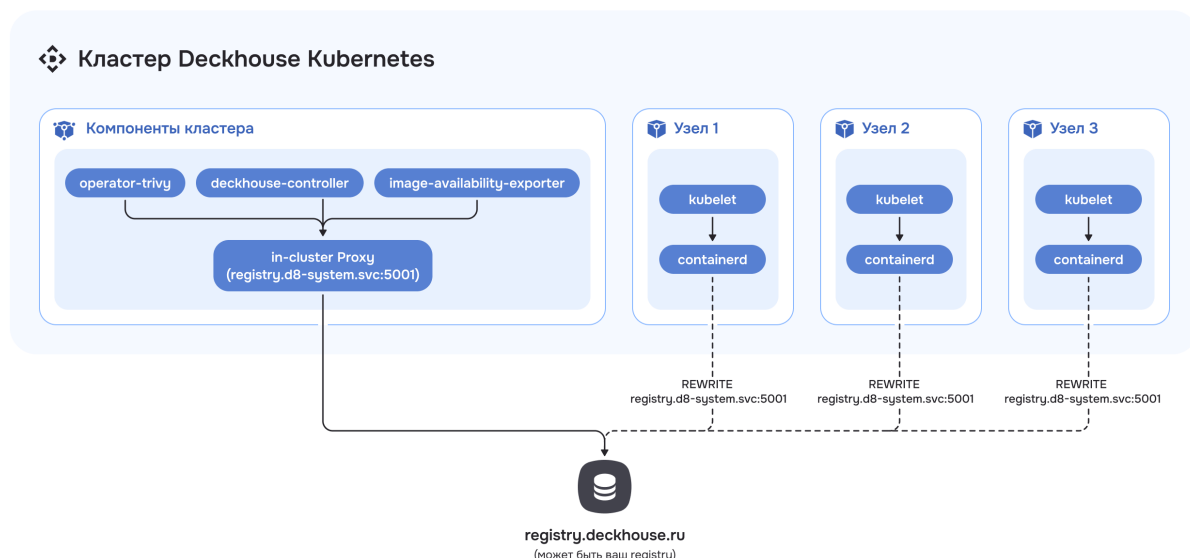
- Изменение параметров registry и переключение режимов доступны только после полного завершения этапа bootstrap.
- При первом переключении необходимо выполнить миграцию пользовательских конфигураций реестра. Подробнее — в разделе «Модуль registry: FAQ».
- Переключение в неконфигурируемый **Unmanaged** режим доступно только из **Unmanaged** режима. Подробнее — в разделе «Модуль registry: FAQ».
- Переключение между режимами **Local** и **Proxy** возможно только через промежуточные режимы **Direct** или **Unmanaged**. Пример последовательности переключения: **Local** / **Proxy** → **Direct** → **Proxy** / **Local**.

Архитектура режима Direct

В режиме **Direct** запросы к registry обрабатываются напрямую, без промежуточного кеширования.

Перенаправление запросов к registry от CRI осуществляется при помощи его настроек, которые указываются в конфигурации `containerd`.

В случае таких компонентов, как `operator-trivy`, `image-availability-exporter`, `deckhouse-controller` и ряда других, обращающихся к registry напрямую, запросы будут идти через in-cluster proxy, расположенный на master-узлах.



Архитектура режима Proxy



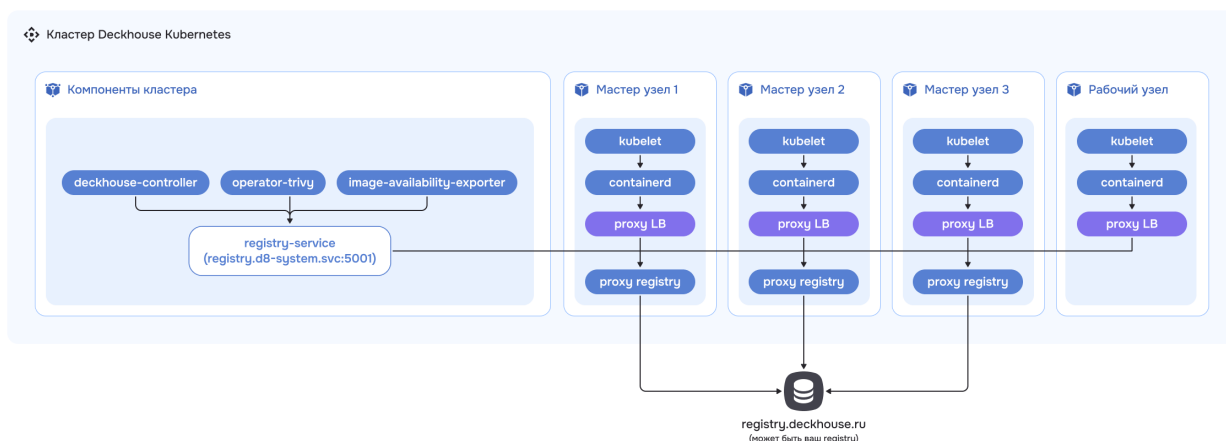
Рекомендуется использовать разные диски для хранения данных registry (`/opt/deckhouse/registry`) и etcd. Использование одного диска может привести к деградации производительности etcd при операциях с registry.

Proxy режим позволяет registry выступать в качестве промежуточного прокси-сервера между клиентом и удалённым registry.

Кеширующий proxy registry запускается в виде статических подов на узлах control-plane (master). Кешируемые данные хранятся на control-plane (master) узлах в директории `/opt/deckhouse/registry`.

Для обеспечения высокой доступности к кеширующему proxy registry используется балансировщик, установленный на каждый узел кластера. CRI обращается к proxy registry через балансировщик. Настройки для обращения к балансировщику указываются в конфигурации `containerd`.

В случае таких компонентов, как `operator-trivy`, `image-availability-exporter`, `deckhouse-controller` и ряда других, обращающихся к registry напрямую, запросы будут идти через кеширующий proxy registry.



Архитектура режима Local



Рекомендуется использовать разные диски для хранения данных registry (/opt/deckhouse/registry) и etcd. Использование одного диска может привести к деградации производительности etcd при операциях с registry.

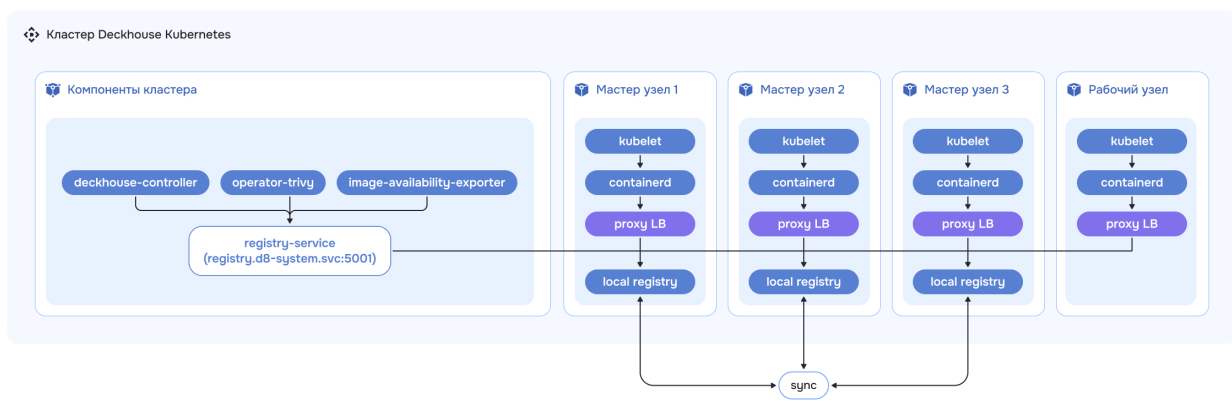
Local режим позволяет создавать локальную копию registry внутри кластера. Образы из удалённого registry полностью скопированы в локальное хранилище и синхронизированы между репликами локального registry.

Работа локального registry идентична работе кеширующего proxy registry. Локальный registry запускается в виде статических подов на узлах control-plane (master). Данные registry хранятся на control-plane (master) узлах в директории /opt/deckhouse/registry.

Для обеспечения высокой доступности к локальному registry используется балансировщик, установленный на каждый узел кластера. CRI обращается к локальному registry через балансировщик. Настройки для обращения к балансировщику указываются в конфигурации containerd.

В случае таких компонентов, как operator-trivy, image-availability-exporter, deckhouse-controller и ряда других, обращающихся к registry напрямую, запросы будут идти в локальный registry.

Локальный registry наполняется с помощью инструмента d8 (команды d8 mirror push/pull). Подробнее — в разделе «Модуль registry: пример использования»



Настройка



Стадия жизненного цикла модуля: Preview



Для настройки параметров подключения к container registry и управления режимом работы с container registry используйте секцию registry конфигурации модуля deckhouse.

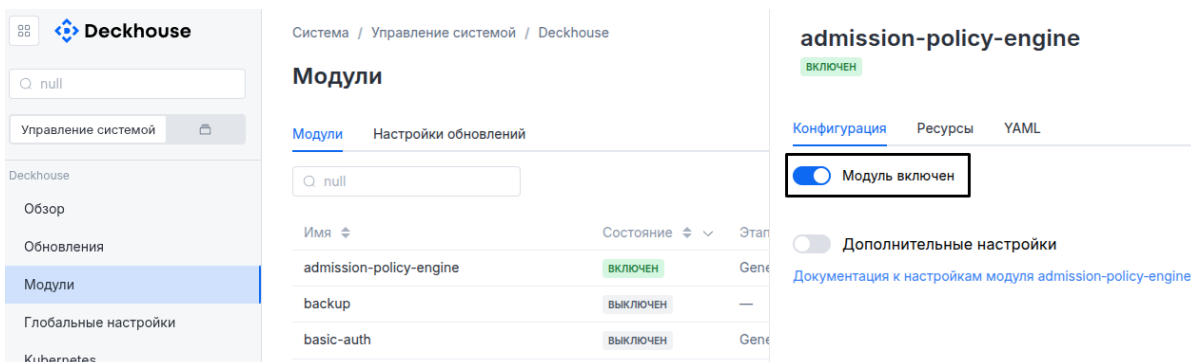
Параметры самого модуля registry указываются в ModuleConfig registry.

Модуль **включен** по умолчанию в наборе модулей Default. Модуль **выключен** по умолчанию в наборах модулей: Managed, Minimal.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль registry, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `registry`:

```
d8 system module enable registry
```

- **С помощью ModuleConfig `registry`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `registry` (создайте его, при необходимости).

Пример манифеста для включения модуля `registry`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: true
```

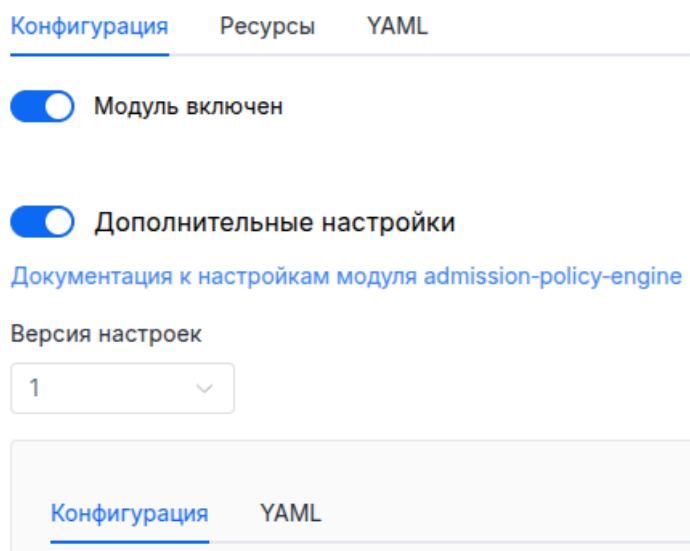
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `registry`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `registry` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `registry`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `registry` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc registry
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `registry`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `registry`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.https**
объект

Тип используемого сертификата.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.https.certManager**

объект

Настройки для certmanager.

■ **settings.https.certManager.clusterIssuerName**

строка

Тип ClusterIssuer'a, используемого для заказа SSL-сертификата (в данный момент доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но возможно определить свои).

По умолчанию: `letsencrypt`

Пример:

```
clusterIssuerName: letsencrypt
```

■ **settings.https.customCertificate**

объект

Настройки для использования пользовательского сертификата.

■ **settings.https.customCertificate.secretName**

строка

Имя секрета в пространстве имен `d8-system`, который будет использовать ingress registry.

Секрет должен быть в формате `kubernetes.io/tls`.

■ **settings.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — доступ по HTTPS с заказом сертификата согласно ClusterIssuer, заданному в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — доступ по HTTPS с использованием сертификата из пространства имен `d8-system`.
- `Disabled` — доступ только по HTTP.
- `OnlyInURI` — доступ по HTTP, подразумевая, что перед ingress-контроллером находится внешний HTTPS-балансировщик, который терминирует HTTPS-трафик. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `CertManager`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

○ **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для registry.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

○ **settings.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено подключение к registry. Если не указано, то разрешено подключение с любых адресов.

Пример:


```
whitelistSourceRanges:
- 10.0.0.0/10
- 192.168.0.0/16
```

■ **Элемент массива**
строка

Шаблон: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\[3[0-2]|[1-2][0-9]|[0-9])?\$`

FAQ



Стадия жизненного цикла модуля: Preview

Как мигрировать на модуль registry?

Во время миграции, для containerd v1 будет выполнен переход на новую схему конфигурации registry. containerd v2 использует новую схему по умолчанию. Подробнее можно ознакомиться в разделе с описанием способов конфигурации

Для containerd v2

1. Выполните переключение на использование модуля `registry`. Для этого, укажите в `moduleConfig deckhouse` параметры `Unmanaged` режима. Если используется `registry`, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля `deckhouse` для корректной настройки.

Посмотреть текущие настройки `registry` можно с помощью команды:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller global values | yq e '.modulesImages.registry' -
```

Данные настройки укажите при конфигурации `Unmanaged` режима:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.ru/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

2. Дождитесь завершения переключения. Пример статуса переключения:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

Для containerd v1

- Во время переключения containerd v1 сервис будет перезапущен.
- Во время переключения containerd v1 будет переведен на новую схему конфигурации registry.
- Во время переключения, пользовательские конфигурации registry для containerd v1 будут временно недоступны.

1. Убедитесь, что на узлах с containerd v1 отсутствуют пользовательские конфигурации registry, расположенные в директории `/etc/containerd/conf.d`.
2. Если конфигурации присутствуют, необходимо выполнить миграцию на новый формат конфигурации registry в containerd. Для этого, необходимо добавить новые конфигурации в директорию `/etc/containerd/registry.d`. Данные конфигурации вступят в силу после переключения на модуль registry. Для добавления конфигураций подготовьте `NodeGroupConfiguration`, подробнее в разделе с описанием способов конфигурации. Пример:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Шаг может быть любой, т.к. не требуется перезапуск сервиса containerd
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/hosts.toml" - <<
EOF
[host]
[host."https://${REGISTRY_URL}"]
  capabilities = ["pull", "resolve"]
[host."https://${REGISTRY_URL}".auth]
  username = "username"
  password = "password"
EOF
```

3. Примените `NodeGroupConfiguration`. Дождитесь появления конфигурационных файлов в директории `/etc/containerd/registry.d` на всех узлах.
4. Проверьте корректность работы конфигураций. Для этого воспользуйтесь командой:

```
# Для https:
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/
private.registry.example/registry/path:tag

# Для http:
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/registry/path:tag
```

5. Выполните переключение на использование модуля `registry`. Для этого, укажите в `moduleConfig deckhouse` параметры `Unmanaged` режима. Если используется `registry`, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля `deckhouse` для корректной настройки.

Посмотреть текущие настройки `registry` можно с помощью команды:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -C deckhouse -- deckhouse-controller global
values | yq e '.modulesImages.registry' -
```

Данные настройки укажите при конфигурации `Unmanaged` режима:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.ru/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

6. После применения, дождитесь в статусе переключения сообщение:

Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "2025-08-13T15:22:34Z"
  message: |
    Check current nodes configuration
    2/2 node(s) Unready:
    - master-0: has custom toml merge containerd configuration
    - worker-5e389be0-578df-s5sm5: has custom toml merge containerd
  configuration
  reason: Processing
  status: "False"
  type: ContainerdConfigPreflightReady
```

Данное сообщение означает, что на узлах имеются старые конфигурации `registry`, расположенные в директории `/etc/containerd/conf.d`. И в данный момент переключение на новую конфигурацию `containerd` заблокировано. Для того, чтобы разрешить переключение, необходимо удалить старые конфигурационные файлы.

7. Удалите старые конфигурационные файлы, чтобы разрешить переключение на модуль `registry`. Для этого создайте `NodeGroupConfiguration`. Пример манифеста `NodeGroupConfiguration`:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth-delete.sh
spec:
  # Шаг должен выполняться до '032_configure_containerd.sh'
  weight: 0
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    file="/etc/containerd/conf.d/old-config.toml"

    [ -f "$file" ] && rm -f "$file"

```

8. После удаления старых конфигураций, убедитесь, что переключение продолжило выполняться. Пример статуса переключения:

```

conditions:
# ...
- lastTransitionTime: "2025-08-13T16:42:09Z"
  message: ""
  reason: ""
  status: "True"
  type: ContainerdConfigPreflightReady

```

9. Дождитесь завершения переключения. Пример статуса переключения:

```

conditions:
# ...
- lastTransitionTime: "..."
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged

```

10. Удалите NodeGroupConfiguration, созданный на шаге удаления старых конфигурационных файлов:

```
d8 k delete nodegroupconfiguration containerd-additional-config-auth-delete.sh
```

Чтобы убедиться, что NodeGroupConfiguration удалён, используйте команду:

```
d8 k get nodegroupconfiguration
```

В списке не должно быть NodeGroupConfiguration, подлежащего удалению (в этом примере — `containerd-additional-config-auth-delete.sh`).

Как мигрировать обратно с модуля registry?



- Это устаревший (deprecated) формат управления registry.
- Во время переключения containerd v1 будет перезапущен.
- Во время переключения containerd v1 будет переведен на старую схему конфигурации registry.
- Во время переключения, пользовательские конфигурации registry для containerd v1 будут временно недоступны.

1. Переведите registry в режим `Unmanaged`. Если используется registry, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля deckhouse для корректной настройки.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.ru/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

2. Проверьте статус переключения, используя инструкцию. Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

3. Переведите registry в неконфигурируемый режим `Unmanaged`. Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
```

4. Проверьте статус переключения, используя инструкцию. Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

5. Если используется containerd v1, и в кластере применены пользовательские конфигурации registry, их необходимо заменить на старый формат. Для этого, подготовьте конфигурации registry старого формата. Данные конфигурации на данном этапе применять не нужно. Пример конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-additional-config-auth.sh
spec:
  # Для добавления файла перед шагом '032_configure_containerd.sh'
  weight: 31
  bundles:
    - '*'
  nodeGroups:
    - "*"
  content: |
    # Copyright 2023 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]

    [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
      endpoint = ["https://${REGISTRY_URL}"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]

    [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
      username = "username"
      password = "password"
      # OR
      auth = "dXNlcm5hbWU6cGFzc3dvcmQ="

    EOF

```

6. Удалите секрет `registry-bashible-config`. Во время удаления, containerd v1 переключится на старый формат конфигурации containerd:

```
d8 k -n d8-system delete secret registry-bashible-config
```

7. После удаления дождитесь завершения переключения. Для отслеживания используйте инструкцию. Пример вывода:

```

conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged

```

8. Если используется containerd v1, примените заготовленные этапом ранее NodeGroupConfiguration с пользовательскими конфигурациями registry.
9. Отключите модуль `registry`. Пример:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: false
  settings: {}
  version: 1

```

Как посмотреть статус переключения режима registry?

Статус переключения режима registry можно получить с помощью следующей команды:

```

d8 k -n d8-system -o yaml get secret registry-state | yq -C -P '.data | del .state |
map_values(@base64d) | .conditions = (.conditions | from_yaml)'

```

Пример вывода:


```

conditions:
- lastTransitionTime: "2025-07-15T12:52:46Z"
  message: 'registry.deckhouse.ru: all 157 items are checked'
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
- lastTransitionTime: "2025-07-11T11:59:03Z"
  message: ""
  reason: ""
  status: "True"
  type: ContainerdConfigPreflightReady
- lastTransitionTime: "2025-07-15T12:47:47Z"
  message: ""
  reason: ""
  status: "True"
  type: TransitionContainerdConfigReady
- lastTransitionTime: "2025-07-15T12:52:48Z"
  message: ""
  reason: ""
  status: "True"
  type: InClusterProxyReady
- lastTransitionTime: "2025-07-15T12:54:53Z"
  message: ""
  reason: ""
  status: "True"
  type: DeckhouseRegistrySwitchReady
- lastTransitionTime: "2025-07-15T12:55:48Z"
  message: ""
  reason: ""
  status: "True"
  type: FinalContainerdConfigReady
- lastTransitionTime: "2025-07-15T12:55:48Z"
  message: ""
  reason: ""
  status: "True"
  type: Ready
mode: Direct
target_mode: Direct

```

Вывод отображает состояние процесса переключения. Каждое условие может находиться в статусе `True` или `False`, а также содержать поле `message` с пояснением.

Описание условий:

Условие	Описание
ContainerdConfigPreflightReady	Состояние проверки конфигурации containerd. Проверяется, что на узлах отсутствуют пользовательские auth конфигурации containerd.
TransitionContainerdConfigReady	Состояние подготовки конфигурации containerd в новый режим. Проверяется, что конфигурация containerd успешно подготовлена и содержит одновременно конфигурации нового и старого режима.
FinalContainerdConfigReady	Состояние завершения переключения containerd в новый режим. Проверяется, что конфигурация containerd успешно применена и содержит конфигурацию нового режима.

Условие	Описание
<code>DeckhouseRegistrySwitchReady</code>	Состояние переключения Deckhouse и его компонентов на использование нового registry. Значение <code>True</code> указывает, что Deckhouse успешно переключился на сконфигурированный registry и готов к работе.
<code>InClusterProxyReady</code>	Состояние готовности In-Cluster Proxy. Проверяется, что In-Cluster Proxy успешно запущен и работает.
<code>CleanupInClusterProxy</code>	Состояние очистки In-Cluster Proxy, если прокси не нужен для работы желаемого режима. Проверяется, что все ресурсы, связанные с In-Cluster Proxy, успешно удалены.
<code>NodeServicesReady</code>	Состояние готовности Node Services Manager и Static-Pod registry. Проверяется, что Node Services Manager успешно запущен и работает, и что Static-Pod registry был успешно развёрнут с помощью Node Services Manager.
<code>CleanupNodeServices</code>	Состояние очистки Node Services Manager и Static-Pod registry, если компоненты не нужны для работы желаемого режима. Проверяется, что все ресурсы, связанные с Node Services Manager и Static-Pod registry, успешно удалены.
<code>RegistryContainsRequiredImages</code>	Состояние проверки registry на наличие необходимых образов.
<code>Ready</code>	Общее состояние готовности registry к работе в указанном режиме. Проверяется, что все предыдущие условия выполнены и модуль готов к работе.

Пример использования



Стадия жизненного цикла модуля: `Preview`



Если в процессе переключения образ какого-либо модуля не загрузился заново и модуль не переустановился, для устранения проблемы воспользуйтесь инструкцией.

Переключение на режим `Direct`

Для переключения уже работающего кластера на режим `Direct` выполните следующие шаги:



При первом переключении с режима `Unmanaged` на режим `Direct` произойдёт полный перезапуск всех компонентов DKP.

1. Перед переключением выполните миграцию на использование модуля `registry`.
2. Убедитесь, что модуль `registry` включен и работает. Для этого выполните следующую команду:

```
d8 k get module registry -O wide
```

Пример вывода:

NAME	WEIGHT	...	PHASE	ENABLED	DISABLED	MESSAGE	READY
registry	38	...	Ready	True			True

3. Убедитесь, что все master-узлы находятся в состоянии `Ready` и не имеют статуса `SchedulingDisabled`, используя следующую команду:

```
d8 k get nodes
```

Пример вывода:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready	control-plane,master	...

Пример вывода, когда master-узел (`master-2` в примере) находится в статусе `SchedulingDisabled`:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready,SchedulingDisabled	control-plane,master	...

4. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 system queue list
```

Пример вывода:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```

5. Установите настройки режима `Direct` в `ModuleConfig deckhouse`. Если используется `registry`, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля `deckhouse` для корректной настройки.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        imagesRepo: registry.deckhouse.ru/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

6. Проверьте статус переключения `registry` в секрете `registry-state`, используя инструкцию.

Пример вывода:

```

conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Direct
target_mode: Direct

```

Переключение на режим Proxy

Для переключения уже работающего кластера на режим Proxy выполните следующие шаги:



- При первом переключении с режима Unmanaged на режим Proxy произойдёт полный перезапуск всех компонентов DKP.
- Переключение из режима Local в Proxy недоступно. Для переключения из режима Local необходимо переключить registry на другой доступный режим (например, Direct).

1. Перед переключением выполните миграцию на использование модуля registry.
2. Убедитесь, что модуль registry включен и работает. Для этого выполните следующую команду:

```
d8 k get module registry -O wide
```

Пример вывода:

NAME	WEIGHT	...	PHASE	ENABLED	DISABLED	MESSAGE	READY
registry	38	...	Ready	True			True

3. Убедитесь, что все master-узлы находятся в состоянии Ready и не имеют статуса SchedulingDisabled, используя следующую команду:

```
d8 k get nodes
```

Пример вывода:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready	control-plane,master	...

Пример вывода, когда master-узел (master-2 в примере) находится в статусе SchedulingDisabled:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready,SchedulingDisabled	control-plane,master	...

4. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 system queue list
```

Пример вывода:

Summary:

- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.

- Установите настройки режима `Proxy` в `ModuleConfig deckhouse`. Если используется `registry`, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля `deckhouse` для корректной настройки.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Proxy
    proxy:
      imagesRepo: registry.deckhouse.ru/deckhouse/ee
      scheme: HTTPS
      license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

- Проверьте статус переключения `registry` в секрете `registry-state`, используя инструкцию.

Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Proxy
target_mode: Proxy
```

Переключение на режим `Local`

Для переключения уже работающего кластера на режим `Local` выполните следующие шаги:



- При первом переключении с режима `Unmanaged` на режим `Local` произойдёт полный перезапуск всех компонентов DKP.
- Переключение из режима `Proxy` в `Local` недоступно. Для переключения из режима `Proxy` необходимо переключить `registry` на другой доступный режим (например, `Direct`).

- Перед переключением выполните миграцию на использование модуля `registry`.
- Убедитесь, что модуль `registry` включен и работает. Для этого выполните следующую команду:

```
d8 k get module registry -O wide
```

Пример вывода:

NAME	WEIGHT	...	PHASE	ENABLED	DISABLED	MESSAGE	READY
registry	38	...	Ready	True			True

3. Убедитесь, что все master-узлы находятся в состоянии `Ready` и не имеют статуса `SchedulingDisabled`, используя следующую команду:

```
d8 k get nodes
```

Пример вывода:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready	control-plane,master	...

Пример вывода, когда master-узел (`master-2` в примере) находится в статусе `SchedulingDisabled`:

NAME	STATUS	ROLES	...
master-0	Ready	control-plane,master	...
master-1	Ready	control-plane,master	...
master-2	Ready,SchedulingDisabled	control-plane,master	...

4. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 system queue list
```

Пример вывода:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```

5. Подготовьте архивы с образами DKP текущей версии. Для этого, воспользуйтесь командой `d8 mirror`.

Пример:

```
TAG=$(
  d8 k -n d8-system get deployment/deckhouse -o yaml \
  | yq -r '.spec.template.spec.containers[] | select(.name == "deckhouse").image |
  split(":")[-1]'
) && echo "TAG: $TAG"

EDITION=$(
  d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller global values -o yaml \
  | yq .deckhouseEdition
) && echo "EDITION: $EDITION"
```

```
d8 mirror pull \
--license="<LICENSE_KEY>" \
--source="registry.deckhouse.ru/deckhouse/$EDITION" \
--deckhouse-tag="$TAG" \
/home/user/d8-bundle
```

6. Установите настройки режима `Local` в ModuleConfig `deckhouse`.

Пример конфигурации:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Local

```

7. Проверьте статус переключения registry в секрете `registry-state`, используя инструкцию. В статусе необходимо дождаться запуска проверки `RegistryContainsRequiredImages`. Условие отобразит отсутствие или наличие образов в запущенном локальном registry.

Пример вывода:

```

conditions:
# ...
- lastTransitionTime: "...
  message: |-
    Mode: Default
    master-1: 0 of 166 items processed, 166 items with errors:
    - source: module/control-plane-manager/control-plane-manager133
      image:
10.128.0.5:5001/system/deckhouse@sha256:00202db19b40930f764edab5695f450cf709d50
736e012055393447b3379414a
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:00202db19b40930f76
4edab5695f450cf709d50736e012055393447b3379414a: unexpected status code 404 Not
Found (HEAD responses have no body, use GET for details)
    - source: module/cloud-provider-yandex/cloud-metrics-exporter
      image:
10.128.0.5:5001/system/deckhouse@sha256:05517a86fcf0ec4a62d14ed7dc4f9ffd91c0571
6b8b0e28263da59edf11f0fad
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:05517a86fcf0ec4a62
d14ed7dc4f9ffd91c05716b8b0ed86d6a1f465f4556fb8: unexpected status code 404 Not
Found (HEAD responses have no body, use GET for details)
    - source: module/control-plane-manager/kube-controller-manager132
      image:
10.128.0.5:5001/system/deckhouse@sha256:13f24cc717698682267ed2b428e7399b145a4d8
ffe96ad1b7a0b3269b17c7e61
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:13f24cc71769868226
7ed2b428e7399b145a4d8ffe96ad1b7a0b3269b17c7e61: unexpected status code 404 Not
Found (HEAD responses have no body, use GET for details)

    ...and more
  reason: Processing
  status: "False"
  type: RegistryContainsRequiredImages

```

8. Загрузите образы в локальный registry с помощью команды `d8 mirror`. Загрузка образов в локальный registry осуществляется через Ingress по адресу `registry.${PUBLIC_DOMAIN}`.

Получите пароль read-write пользователя локального registry:

```
$ d8 k -n d8-system get secret/registry-user-rw -o json | jq -r '.data | to_entries[] | "\n(.key): \(.value | @base64d)"'
name: rw
password: KfVxXZGuqKkkumPz
passwordHash: $2a$10$Phjbr6iinLf00ZZDD2Y70.p9H3nD0gYzFmpYKW5eydGvIsdaHQY0a
```

Загрузите образы в локальный registry:

```
d8 mirror push \
--registry-login="rw" \
--registry-password="KfVxXZGuqKkkumPz" \
/home/user/d8-bundle \
registry.${PUBLIC_DOMAIN}/system/deckhouse
```

- Проверьте статус переключения registry в секрете `registry-state`, используя инструкцию. После загрузки образов статус `RegistryContainsRequiredImages` должен быть в состоянии `Ready`

Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: |-
    Mode: Default
    master-1: all 166 items are checked
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
hash: ..
mode: Direct
target_mode: Local
```

- Дождитесь завершения переключения. Для проверки статуса переключения воспользуйтесь инструкцией.

Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Local
target_mode: Local
```

Переключение на режим `Unmanaged`

Для переключения уже работающего кластера на режим `Unmanaged` выполните следующие шаги:



Изменение registry в `Unmanaged` режиме приведёт к перезапуску всех компонентов DKP.

- Перед переключением выполните миграцию на использование модуля `registry`.
- Убедитесь, что модуль `registry` включен и работает. Для этого выполните следующую команду:


```
d8 k get module registry -O wide
```

Пример вывода:

NAME	WEIGHT	...	PHASE	ENABLED	DISABLED	MESSAGE	READY
registry	38	...	Ready	True			True

3. Проверьте, чтобы очередь Deckhouse была пустой и без ошибок:

```
d8 system queue list
```

Пример вывода:

```
Summary:
- 'main' queue: empty.
- 107 other queues (0 active, 107 empty): 0 tasks.
- no tasks to handle.
```

4. Установите настройки режима `Unmanaged` в `ModuleConfig deckhouse`. Если используется `registry`, отличный от `registry.deckhouse.ru`, ознакомьтесь с конфигурацией модуля `deckhouse` для корректной настройки.

Пример конфигурации:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: registry.deckhouse.ru/deckhouse/ee
        scheme: HTTPS
        license: <LICENSE_KEY> # Замените на ваш лицензионный ключ
```

5. Проверьте статус переключения `registry` в секрете `registry-state`, используя инструкцию.

Пример вывода:

```
conditions:
# ...
- lastTransitionTime: "...
  message: ""
  reason: ""
  status: "True"
  type: Ready
hash: ..
mode: Unmanaged
target_mode: Unmanaged
```

6. При необходимости переключения на старый метод управления `registry`, ознакомьтесь с инструкцией.



Это устаревший (deprecated) формат управления `registry`.

Модуль `registry-packages-proxy`

Модуль `registry-packages-proxy` предоставляет сервис HTTP-прокси внутри кластера для доступа к пакетам из хранилищ образов контейнеров (container registry). Он выступает в качестве посредника между компонентами кластера и внешними или внутренними хранилищами образов контейнеров с функциями кеширования для оптимизации использования пропускной способности сети и повышения производительности при загрузке пакетов.

Этот модуль — критически важный компонент инфраструктуры, который работает на master-узлах и используется во время загрузки кластера и операций в процессе работы кластера для извлечения пакетов из хранилищ образов контейнеров.

Модуль развёртывает высокодоступный прокси-сервис, который:

- Работает как deployment на master-узлах с включенным `hostNetwork` для обеспечения доступности во время загрузки, когда CNI ещё недоступен.
- Прослушивает порт 4219 (HTTPS) на IP-адресе каждого master-узла.
- Предоставляет эндпоинт `/package` для извлечения пакетов из хранилища образов контейнеров по дайджесту.
- Реализует локальное кеширование извлечённых пакетов (до 1 ГБ) для снижения сетевого трафика и улучшения производительности.
- Следит за кастомными ресурсами ModuleSource для получения учётных данных хранилищ образов контейнеров.
- Использует `kube-rbac-proxy` для защиты доступа к прокси и эндпоинтам метрик.
- Предоставляет публичный HTTPS API (через Ingress) для исполняемых файлов и плагинов Deckhouse CLI.
- Предоставляет внутрикластерный HTTPS API для иконок пакетов (без публикации через Ingress).

Архитектура

Прокси-сервис состоит из двух контейнеров:

1. **registry-packages-proxy** — основное приложение прокси, которое:
 - извлекает пакеты из удалённых хранилищ образов контейнеров с использованием дайджестов;
 - кеширует пакеты локально в эфемерном volume (максимум 1 ГБ);
 - поддерживает аутентификацию в хранилищах образов контейнеров через учётные данные из ресурсов ModuleSource;
 - предоставляет проверки работоспособности и метрики Prometheus;
 - прослушивает 127.0.0.1:5080 (HTTP, внутренний).
2. **kube-rbac-proxy** — обеспечивает контроль доступа на основе RBAC:
 - предоставляет сервис на порту 4219 (HTTPS);
 - защищает эндпоинт `/metrics` с авторизацией Kubernetes RBAC;
 - защищает эндпоинт `/package`, требуя соответствующих разрешений;
 - защищает `/v1/images/*` (загрузка Deckhouse CLI) с авторизацией Kubernetes RBAC;
 - позволяет доступ к `/healthz` без аутентификации.

HTTP API

После завершения развёртывания кластера (bootstrap) и настройки шаблона DNS-имен в параметре `publicDomainTemplate`, модуль создаёт Ingress для имени `registry-packages-proxy`, используя шаблон из `publicDomainTemplate` (например, `registry-packages-proxy.company.my`, для `publicDomainTemplate: "%s.company.my"`).

Эндпоинты ниже доступны двумя способами:

- **Публично** (через Ingress на публичном имени `registry-packages-proxy`): только `/v1/images/*`.
- **Только внутри кластера** (через Service `registry-packages-proxy.d8-cloud-instance-manager.svc` на порту 443, либо через `:4219` на каждом master-узле в фазе bootstrap): все маршруты, включая `/v1/packages/*`.

Иконки пакетов (`/v1/packages/`) — только внутри кластера

Иконки пакетов отдаются **без аутентификации** (`kube-rbac-proxy` исключает эти пути из RBAC), но **доступны только внутри кластера**. Маршрут `/v1/packages/*` намеренно **не публикуется через Ingress**, поэтому публичное имя `registry-packages-proxy.<publicDomain>` иконки не отдаёт.

Метод	Путь	Описание
-------	------	----------

Метод	Путь	Описание
GET, HEAD	/v1/packages/<РЕПОЗИТОРИЙ-ПАКЕТОВ>/<ИМЯ-ПАКЕТА>/metadata/icon/	Иконка последнего semver-тега
GET, HEAD	/v1/packages/<РЕПОЗИТОРИЙ-ПАКЕТОВ>/<ИМЯ-ПАКЕТА>/metadata/icon	То же, что выше
GET, HEAD	/v1/packages/<РЕПОЗИТОРИЙ-ПАКЕТОВ>/<ИМЯ-ПАКЕТА>/metadata/icon/<ВЕРСИЯ>	Иконка указанной версии (<ВЕРСИЯ> — semver, например v1.0.1)

<РЕПОЗИТОРИЙ-ПАКЕТОВ> — это `metadata.name` кастомного ресурса `PackageRepository`; поле `spec.registry.геро` этого ресурса определяет, по какому пути в хранилище образов прокси читает иконку. Прокси читает иконку из OCI-образа `<spec.registry.геро>/<ИМЯ-ПАКЕТА>:<ТЕГ>`.

Прокси ищет внутри образа пакета следующие файлы в указанном порядке приоритета и возвращает первый найденный:

Путь	Content-Type
docs/icon.svg	image/svg+xml
docs/icon.png	image/png
docs/icon.jpg	image/jpeg
docs/icon.jpeg	image/jpeg

Если ни одного из этих файлов нет (или файл больше 4 МиБ), прокси возвращает `404 Not Found`; вызывающий код должен использовать иконку по умолчанию. SVG предпочтительнее, так как не зависит от разрешения.

Пример запроса из пода внутри кластера:

```
curl -fsSk "https://registry-packages-proxy.d8-cloud-instance-manager.svc/v1/packages/my-repo/my-module/metadata/icon/"
```

Пример успешного ответа (когда в образе есть `docs/icon.svg`):

```
Content-Type: image/svg+xml
Content-Disposition: attachment; filename="<ИМЯ-ПАКЕТА>.svg"
```

Загрузка Deckhouse CLI (/v1/images/) — публичный

Эти эндпоинты доступны через публичный Ingress (`registry-packages-proxy.<publicDomain>`) и требуют действительный токен Kubernetes (или клиентский сертификат, принимаемый `kubernetes-proxy`) и право RBAC `get` на `subresource deployments/cli-binary` с именем `registry-packages-proxy` в namespace `d8-cloud-instance-manager`.

Выдайте доступ через ClusterRole `d8:registry-packages-proxy:cli-download` (привяжите к пользователям или ServiceAccount через ClusterRoleBinding или RoleBinding).

Метод	Путь	Описание
GET	/v1/images/<ОБРАЗ>/tags	JSON со списком тегов
GET, HEAD	/v1/images/<ОБРАЗ>/tags/<ТЕГ>	OCI-образ в формате application/x-gzip (слои сведены)

Допустимые значения для <ОБРАЗ>:

- `deckhouse-cli`
- `deckhouse-cli/plugins/<ПЛАГИН>` (один сегмент пути для <ПЛАГИН>)

Пример:

```
curl -fsS -H "Authorization: Bearer ${TOKEN}" "https://registry-packages-proxy.example.com/v1/images/deckhouse-cli/tags"
```

Внутренний эндпоинт `/package`

Устаревший эндпоинт `/package?digest=...` (bootstrap и внутренние компоненты) по-прежнему защищён RBAC (subresource deployments/http). Через публичный Ingress не публикуется.

Поток извлечения пакетов

Когда компонент запрашивает пакет:

1. Запрос включает параметр `digest` (обязательный) и необязательные параметры `repository` и `path`.
2. Прокси проверяет локальный кеш на наличие запрошенного дайджеста.
3. Если запрашиваемый пакет есть в кеше, он извлекается из кеша.
4. Если запрашиваемый пакет отсутствует в кеше:
 - Прокси извлекает учётные данные для указанного хранилища образов из отслеживаемых ресурсов ModuleSource.
 - Пакет извлекается из удалённого хранилища образов.
 - Пакет передаётся клиенту с одновременным кешированием для будущих запросов.
5. Ответы включают соответствующие HTTP-заголовки для кеширования (`Cache-Control` , `ETag` , `Content-Length`).

Высокая доступность

Модуль обеспечивает высокую доступность через:

- Запуск нескольких реплик на master-узлах (в HA-конфигурациях).
- Правила anti-affinity для распределения подов по разным master-узлам.
- PodDisruptionBudget для предотвращения одновременного нарушения работы всех реплик.
- Поддержку Vertical Pod Autoscaler для автоматической настройки ресурсов.

Роли RBAC, создаваемые модулем

ClusterRole	Назначение
<code>d8:registry-packages-proxy:cli-download</code>	Доступ к <code>/v1/images/*</code>
<code>d8:registry-packages-proxy:packages-download</code>	Зарезервирована для будущих защищённых маршрутов <code>/v1/packages/*</code> (иконки отдаются анонимно, доступ только изнутри кластера)

Ограничения

- Модуль работает исключительно на master-узлах.
- Требуется `hostNetwork: true` для работы во время фазы загрузки.
- Размер кеша ограничен 1 ГБ на под.
- Большинство HTTP-эндпоинтов требуют RBAC Kubernetes; без аутентификации доступны только проверки работоспособности (health check) и иконки пакетов. Иконки пакетов дополнительно ограничены доступом только изнутри кластера (без маршрута через публичный Ingress).
- Иконки читаются из фиксированных путей внутри образа пакета (`docs/icon.svg` , `docs/icon.png` , `docs/icon.jpg` , `docs/icon.jpeg`); SVG предпочтительнее. Максимальный размер иконки — 4 МиБ.

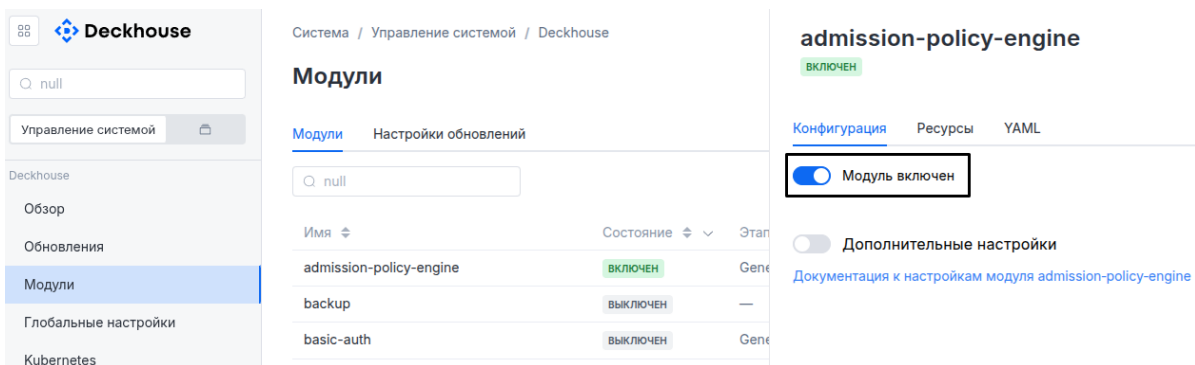
Настройки

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed` , `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `registry-packages-proxy`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `registry-packages-proxy`:

```
d8 system module enable registry-packages-proxy
```

- **С помощью ModuleConfig `registry-packages-proxy`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `registry-packages-proxy` (создайте его, при необходимости).

Пример манифеста для включения модуля `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  enabled: true
```

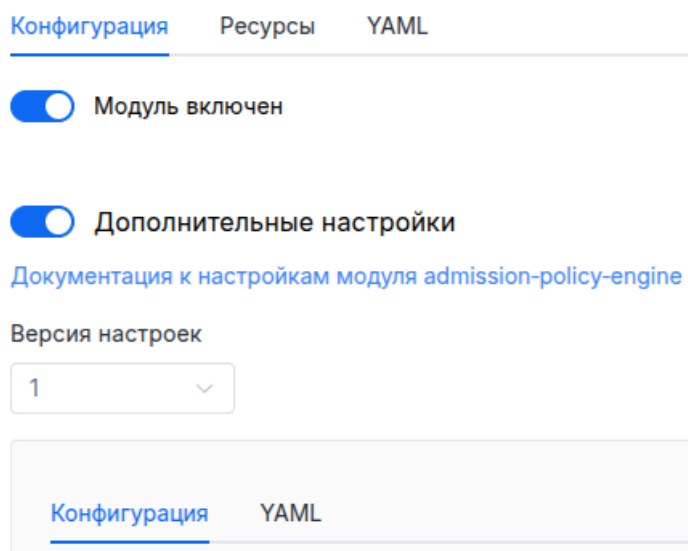
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `registry-packages-proxy`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `registry-packages-proxy` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `registry-packages-proxy`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `registry-packages-proxy` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc registry-packages-proxy
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `registry-packages-proxy`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Модуль не имеет настроек.

Модуль secret-copier

Этот модуль отвечает за копирование секретов во все пространства имён.

Он полезен тем, что позволяет не копировать каждый раз в CI секреты для пуллинга образов и заказа RBD в Ceph.



Модуль `secret-copier` не может использоваться совместно с модулем `multitenancy-manager`.

`multitenancy-manager` создаёт изолированные окружения для пользователей в их проектах, а `secret-copier` автоматически раздаёт секреты во все пространства имён. Если в закрытом окружении пользователя окажутся чувствительные данные, это может привести к утечке данных и нарушению модели безопасности.

Поэтому, если необходимо предоставить общий сертификат (например, WC-сертификат для внутреннего окружения) или общий токен доступа к registry, не используйте `secret-copier`. Поместите такие секреты в шаблон проекта в `multitenancy-manager` - администратор кластера должен определить их в конфигурации проекта.

Как работает?

Модуль `secret-copier` следит за изменениями секретов в пространстве имён `default` с лейблом `secret-copier.deckhouse.io/enabled: ""`.

- Созданный секрет будет скопирован во все пространства имён.
- Изменённый секрет с его новым содержимым будет раскопирован во все пространства имён.
- При удалении секрет будет удален из всех пространств имён.

- При изменении скопированного секрета в прикладном пространстве имён, тот будет перезаписан оригинальным содержимым.
- При создании любого пространства имён в него копируются все секреты из пространства имён `default` с лейблом `secret-copier.deckhouse.io/enabled: ""`.

Кроме этого, каждую ночь секреты будут повторно синхронизированы и приведены к состоянию в пространстве имён `default`.

Что нужно настроить?

Чтобы все заработало, достаточно создать в пространстве имён `default` секрет с лейблом `secret-copier.deckhouse.io/enabled: ""`.

i **Внимание.** Рабочим пространством имён для модуля является `default`, Секреты будут копироваться только из него. Ресурсы с лейблом `secret-copier.deckhouse.io/enabled: ""`, созданные в других пространствах имён при включенном модуле будут автоматически удалены.

Как ограничить список пространств имён, в которые будет производиться копирование?

Для этого нужно задать label-селектор в значении аннотации `secret-copier.deckhouse.io/target-namespace-selector`. Например: `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. Модуль создаст копию этого секрета во всех пространствах имён, соответствующих заданному label-селектору.

Модуль service-with-healthchecks

Ограничения стандартного балансировщика Service

В Kubernetes за внутреннюю и внешнюю балансировку запросов отвечает ресурс типа `Service`. Он распределяет запросы между рабочими подами приложения и исключает из балансировки повреждённые экземпляры. Для проверки способности пода обрабатывать входящие запросы применяются `readiness`-пробы, которые указываются в спецификации контейнеров, входящих в этот под.

Стандартный инструмент балансировки `Service` подходит для большинства задач облачных приложений, но имеет два ограничения:

- Если хотя бы один контейнер в поде не проходит проверку готовности (`readiness`-пробу), весь под отмечается как `NotReady` и исключается из балансировки всех сервисов, с которыми он связан.
- Для каждого контейнера можно настроить только одну пробу, поэтому невозможно создать отдельные пробы для проверки, например, доступности чтения и записи.

Примеры сценариев, где стандартного балансировщика недостаточно:

- База данных:
 - Работает в трёх подах — `db-0`, `db-1` и `db-2`, каждый из которых содержит один контейнер с запущенным процессом базы данных.
 - Необходимо создать два сервиса (`Service`) — `db-write` для записи и `db-read` для чтения.
 - Запросы на чтение должны балансироваться между всеми подами.
 - Запросы на запись балансируются только на тот под, который назначен мастером средствами самой базы данных.
- Виртуальная машина:
 - Под содержит единственный контейнер, в котором запущен процесс `qemu`, выполняющий роль гипервизора для гостевой виртуальной машины.
 - В гостевой виртуальной машине запущены независимые процессы, например, веб-сервер и SMTP-сервер.
 - Требуется создать два `Service` — `web` и `smtp`, каждый из которых будет иметь свои `readiness`-пробы.

Возможности балансировщика ServiceWithHealthcheck

В отличие от стандартного балансировщика, где `readiness`-пробы привязаны к состоянию контейнеров, `ServiceWithHealthcheck` позволяет настраивать активные пробы на отдельные TCP-порты. Таким образом, каждый балансировщик, обслуживающий один и тот же под, может работать независимо от других.

Настроить данный способ балансировки можно при помощи ресурса `ServiceWithHealthchecks`:

- Его спецификация идентична стандартному `Service` с добавлением раздела `healthcheck`, который содержит набор проверок.
- На данный момент поддерживается три вида проб:
 - `TCP` — обычная проверка с помощью установки TCP-соединения.
 - `HTTP` — возможность отправить HTTP-запрос и ожидать определённый код ответа.
 - `PostgreSQL` — возможность отправить SQL-запрос и ожидать его успешного завершения.

Ознакомиться с примерами можно в документации.

Внутреннее устройство балансировщика `ServiceWithHealthcheck`

Балансировщик состоит из двух компонентов:

- контроллер — работает на мастер-узлах кластера и управляет ресурсами `ServiceWithHealthcheck`,
- агенты — работают на каждом узле кластера и выполняют пробы для подов, запущенных на этом узле.

Балансировщик `ServiceWithHealthcheck` спроектирован так, чтобы не зависеть от реализации CNI, используя при этом стандартные ресурсы `Service` и `EndpointSlice`:

- Контроллер при создании ресурса `ServiceWithHealthcheck` автоматически создает одноименный ресурс `Service` в том же пространстве имен с пустым полем `selector`. Это позволяет избежать создания стандартным контроллером `EndpointSlice`, которые используются для настройки балансировки.
- Каждый агент при появлении на своём узле подов, которые попадают под управление `ServiceWithHealthcheck`, осуществляет настроенные пробы и создаёт для них `EndpointSlice` со списком проверенных IP-адресов и портов. Данный `EndpointSlice` привязан к дочернему ресурсу `Service`, созданному выше.
- CNI сопоставит все `EndpointSlice` со стандартными сервисами, созданными выше и осуществит балансировку по проверенным IP-адресам и портам на всех узлах кластера.

Миграция с `Service` на ресурс `ServiceWithHealthchecks`, например в рамках CI/CD, не должна вызвать затруднений. Спецификация `ServiceWithHealthchecks` в основе своей повторяет спецификацию `Service`, но содержит дополнительный раздел `healthchecks`. Во время жизненного цикла ресурса `ServiceWithHealthchecks` создается одноименный сервис в том же namespace, чтобы привычным способом (`kube-proxu` или `cni`) направить трафик на рабочие нагрузки в кластере.

Настройки



Чтобы создаваемые балансировщики ServiceWithHealthchecks работали, необходимо выполнение следующих условий:

- В сетевой политике пользовательского проекта, в котором будет создаваться ServiceWithHealthchecks, должно присутствовать правило, разрешающее входящий трафик из всех подов неймспейса `d8-service-with-healthchecks`:

```
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: d8-service-with-healthchecks
```

Подробнее о сетевых политиках — в разделе «Настройка сетевых политик».

- Кластерная роль, которая используется в ClusterRoleBinding и RoleBinding при назначении прав пользователям и сервисным аккаунтам, для ресурса ServiceWithHealthchecks должна быть расширена следующими правилами:
 - get
 - list
 - watch
 - create
 - update
 - patch
 - delete.

Подробнее — в разделе «Выдача прав пользователям и сервисным аккаунтам».



После включения модуля не происходит автоматическая замена имеющихся ресурсов типа Service на ServiceWithHealthcheck. Для замены имеющихся сервисов на использование ServiceWithHealthcheck выполните следующие действия:

- Создайте ресурсы ServiceWithHealthcheck с такими же именами и параметрами, как существующие ресурсы Service, которые нужно заменить. При создании ServiceWithHealthcheck укажите обязательные параметры `healthchecks`.
- Удалите ресурсы Service, которые требуется заменить ServiceWithHealthcheck.

Модуль не включен по умолчанию в каком-либо наборе модулей.

Явно включить или выключить модуль можно одним из следующих способов:

- С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `service-with-healthchecks`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:

The screenshot shows the Deckhouse web interface. On the left is a sidebar with navigation links: Обзор, Обновления, Модули (selected), Глобальные настройки, and Kubernetes. The main content area is titled 'Модули' and contains a table of modules. The table has columns for 'Имя' (Name), 'Состояние' (Status), and 'Этап' (Stage). The 'admission-policy-engine' module is highlighted, and its status is 'включен' (enabled). Below the table, there is a section for the 'admission-policy-engine' module, showing a toggle switch for 'Модуль включен' (Module enabled) which is currently turned on. There is also a link to 'Документация к настройкам модуля admission-policy-engine'.

- С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `service-with-healthchecks`:

```
d8 system module enable service-with-healthchecks
```

- **С помощью ModuleConfig service-with-healthchecks.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `service-with-healthchecks` (создайте его, при необходимости).

Пример манифеста для включения модуля `service-with-healthchecks`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: service-with-healthchecks
spec:
  enabled: true
```

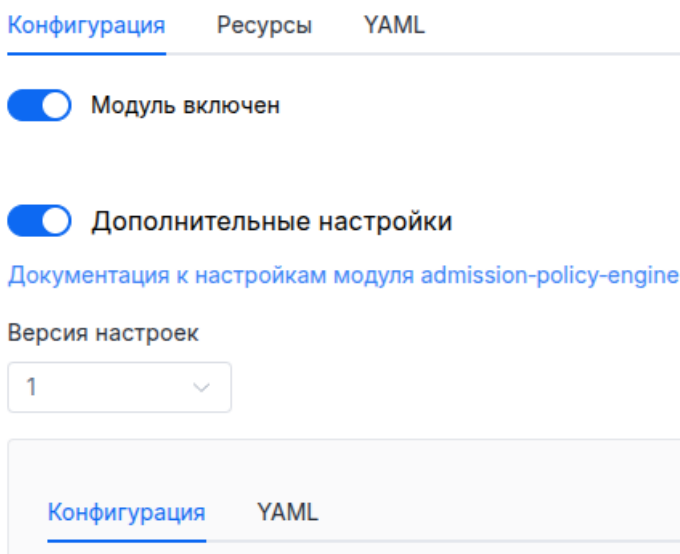
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `service-with-healthchecks`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `service-with-healthchecks` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `service-with-healthchecks`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `service-with-healthchecks` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc service-with-healthchecks
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `service-with-healthchecks`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `service-with-healthchecks`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: service-with-healthchecks
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.debug**
булевый

Включает уровень логирования debug для контроллера и агента, обеспечивая более подробный вывод логов, включая информацию о вызывающем коде.

По умолчанию: `false`

- **settings.verboseStatus**
булевый

Если true, агент будет обновлять поле lastProbeTime в статусе ServiceWithHealthchecks при каждой проверке (каждые periodSeconds). Это может быть полезно для отладки, но создает высокую нагрузку на API-сервер. Если false, фиксируются только изменения состояния.

По умолчанию: `false`

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

ServiceWithHealthchecks

Short names: swh

Scope: Namespaced

Version: v1alpha1

Модуль ServiceWithHealthchecks

- **apiVersion**
строка
- **kind**
строка
- **metadata**
объект
- **spec**
объект

Описывает желаемое состояние ServiceWithHealthchecks.

Чтобы создаваемые балансировщики ServiceWithHealthchecks работали, необходимо выполнение следующих условий:

- В сетевой политике пользовательского проекта, в котором будет создаваться

ServiceWithHealthchecks, должно присутствовать правило, разрешающее входящий трафик из всех подов неймспейса `d8-service-with-healthchecks`:

```
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: d8-service-with-healthchecks
```

Подробнее о сетевых политиках — в разделе «Настройка сетевых политик».

- Кластерная роль, которая используется в ClusterRoleBinding и RoleBinding при назначении прав пользователям и сервисным аккаунтам, для ресурса ServiceWithHealthchecks должна быть расширена следующими правилами:

- `get`
- `list`
- `watch`
- `create`
- `update`
- `patch`
- `delete`.

Подробнее — в разделе «Выдача прав пользователям и сервисным аккаунтам».

- **spec.allocateLoadBalancerNodePorts**
булевый

Определяет, следует ли автоматически выделять NodePort для сервисов типа LoadBalancer. Значение по умолчанию — `true`. Значение можно установить в `false`, если балансировщик нагрузки кластера не зависит от NodePorts.

Если запрашивающая сторона указывает конкретные значения NodePort, эти запросы будут выполнены независимо от значения этого параметра. Данный параметр может быть установлен только для сервисов типа LoadBalancer и автоматически сбрасывается при изменении типа на другой.

- **spec.clusterIP**
строка

IP-адрес сервиса.

Обычно, IP-адрес назначается сервису случайным образом. Если IP-адрес указывается вручную, он должен находиться в допустимом диапазоне (заданном конфигурацией системы) и быть свободным для назначения; в противном случае создание сервиса завершится ошибкой.

Поле не подлежит изменению при обновлении, за исключением случаев, когда тип сервиса также изменяется на `ExternalName` (в этом случае поле должно быть пустым), или когда тип меняется с `ExternalName`, позволяя задать это поле дополнительно, как указано выше.

Допустимые значения: `None`, пустая строка (`""`), или действительный IP-адрес. Установка этого значения в `None` создает сервис без виртуального IP, что полезно, когда предпочтительнее прямое подключение к конечным точкам, и проксирование не требуется. Применимо только к типам ClusterIP, NodePort и LoadBalancer. Если это поле указано при создании сервиса типа `ExternalName`, создание завершится неудачей. Подробнее.

- **spec.externalIPs**
массив строк

Список IP-адресов, по которым узлы в кластере также будут принимать трафик для этого сервиса. Эти IP-адреса не управляются Kubernetes. Пользователь несет ответственность за обеспечение доставки трафика на узел с этим IP-адресом.

Общий пример — внешние балансировщики нагрузки, которые не являются частью системы Kubernetes.

- **spec.externalName**
строка

Внешняя ссылка, которую механизмы обнаружения будут возвращать в качестве псевдонима для этого сервиса (например, DNS-запись CNAME). Проксирование не

будет использоваться. Имя должно быть в нижнем регистре и соответствовать стандарту RFC-1123 и требует, чтобы поле `type` было установлено в `ExternalName`.

- **spec.externalTrafficPolicy**
строка

Описывает, как узлы распределяют трафик сервиса, который они получают через один из его “внешне направленных” адресов (NodePorts, ExternalIPs и LoadBalancer IPs). Если установлено значение `Local`, прокси настроит сервис таким образом, что внешние балансировщики нагрузки будут управлять распределением трафика между узлами, и каждый узел будет передавать трафик только локальным конечным точкам сервиса, не маскируя исходный IP-адрес клиента. Трафик, ошибочно отправленный на узел без конечных точек, будет отброшен.

Значение по умолчанию — `Cluster`, при котором используется стандартное поведение равномерной маршрутизации ко всем конечным точкам (возможно, с модификациями на основе топологии и других особенностей).

Обратите внимание, что трафик, отправленный на External IP или LoadBalancer IP изнутри кластера, всегда будет использовать семантику `Cluster`, но клиенты, отправляющие трафик на NodePort внутри кластера, могут учитывать политику трафика при выборе узла.

- **spec.healthCheckNodePort**
целочисленный

Указывает NodePort для проверки состояния сервиса. Это применимо только тогда, когда тип сервиса установлен на `LoadBalancer`, а параметр `externalTrafficPolicy` установлен на `Local`. Если указано значение, оно находится в допустимом диапазоне и не используется, то оно будет применено.

Если не указано, значение будет назначено автоматически. Внешние системы (например, балансировщики нагрузки) могут использовать этот порт, чтобы определить, содержит ли узел конечные точки для этого сервиса.

Если это поле указано при создании сервиса, который в этом не нуждается, создание завершится ошибкой. Это поле будет очищено при изменении параметров сервиса, например, при изменении типа.

Это поле не может быть обновлено после установки.

- **spec.healthcheck**
объект

Обязательный параметр

- **spec.healthcheck.initialDelaySeconds**
целочисленный

Время в секундах после старта пода, по истечении которого запускаются пробы.

По умолчанию: `10`

Допустимые значения: `0 <= X <= 108000`

- **spec.healthcheck.periodSeconds**
целочисленный

Период в секундах для повторных запусков проб.

По умолчанию: `10`

Допустимые значения: `1 <= X <= 108000`

- **spec.healthcheck.probes**
массив объектов

- **spec.healthcheck.probes.failureThreshold**
целочисленный

Минимальное количество последовательных неудач, после которых проба будет считаться неудачной.

По умолчанию: `3`

Допустимые значения: `1 <= X`

- **spec.healthcheck.probes.http**
объект

- **spec.healthcheck.probes.http.caCert**

строка

CA-сертификат для HTTPS соединения.

- **spec.healthcheck.probes.http.host**

строка

Имя хоста для подключения, по умолчанию используется IP-адрес пода.

- **spec.healthcheck.probes.http.httpHeaders**

массив объектов

Пользовательские заголовки, которые будут добавлены к HTTP-запросу и использованы при выполнении пробы. Разрешены повторяющиеся заголовки.

- **spec.healthcheck.probes.http.httpHeaders.name**

строка

Обязательный параметр

Имя заголовка.

При выводе будет приведено к стандартному виду (lowercase), так что заголовки с разным регистром будут распознаваться как одинаковые.

- **spec.healthcheck.probes.http.httpHeaders.value**

строка

Обязательный параметр

Значение заголовка.

- **spec.healthcheck.probes.http.insecureSkipTLSVerify**

булевый

Указывает, следует ли клиенту пропустить проверку цепочки сертификатов TLS сервера и имени хоста.

По умолчанию: `false`

- **spec.healthcheck.probes.http.method**

строка

Метод, который будет использоваться для запроса.

По умолчанию: `GET`

Допустимые значения: `GET`, `HEAD`

- **spec.healthcheck.probes.http.path**

строка

Путь для доступа к HTTP-серверу.

- **spec.healthcheck.probes.http.scheme**

строка

Схема, используемая для подключения к хосту.

По умолчанию: `HTTP`

Допустимые значения: `HTTP`, `HTTPS`

- **spec.healthcheck.probes.http.targetPort**

целочисленный

Обязательный параметр

Номер порта для доступа к контейнеру.

Допустимые значения: `1 <= X <= 65535`

- **spec.healthcheck.probes.mode**

строка

Режим для проверки.

Допустимые значения: `HTTP`, `TCP`, `PostgreSQL`

■ **spec.healthcheck.probes.postgreSQL**

объект

■ **spec.healthcheck.probes.postgreSQL.authSecretName**

строка

Имя секрета (Secret), расположенного в том же пространстве имен, что и этот ресурс. Секрет может содержать следующие поля аутентификации: `tlsMode`, `clientCert`, `clientKey`, `aCert`, `password`, `user`.

i

Внимание. Тип секрета должен быть `network.deckhouse.io/postgresql-credentials`.

■ **spec.healthcheck.probes.postgreSQL.dbName**

строка

Имя базы данных, используемое при подключении. Если не указать, то будет использовано имя пользователя.

■ **spec.healthcheck.probes.postgreSQL.query**

строка

Запрос к PostgreSQL для тестирования.

Чтобы считаться успешным, запрос должен возвращать одно из следующих значений: `TRUE`, `t`, `true`, `y`, `yes`, `1`.

i

Внимание. Привилегии для пользователя, под которым выполняются запросы, должны быть минимальными.

По умолчанию: `select 1`

■ **spec.healthcheck.probes.postgreSQL.targetPort**

целочисленный

Обязательный параметр

Номер порта для доступа к контейнеру.

Допустимые значения: `1 <= X <= 65535`

■ **spec.healthcheck.probes.successThreshold**

целочисленный

Минимальное количество последовательных успешных проверок, после которых проверка считается успешной после предыдущих неудач.

По умолчанию: `1`

Допустимые значения: `1 <= X`

■ **spec.healthcheck.probes.tcp**

объект

■ **spec.healthcheck.probes.tcp.targetPort**

целочисленный

Обязательный параметр

Номер порта для доступа к контейнеру.

Допустимые значения: `1 <= X <= 65535`

■ **spec.healthcheck.probes.timeoutSeconds**

целочисленный

Число секунд, по истечении которых проверка завершится по таймауту.

По умолчанию: `1`

Допустимые значения: `1 <= X`

- **spec.internalTrafficPolicy**

строка

Описывает, как узлы распределяют трафик сервиса, который они получают через ClusterIP.

Если параметр установлен на `Local`, прокси предполагает, что поды взаимодействуют только с конечными точками сервиса, находящимися на том же узле, что и под. Трафик будет отброшен, если на узле нет локальных конечных точек.

Значение по умолчанию — `Cluster`, при котором используется стандартное поведение маршрутизации ко всем конечным точкам равномерно (возможны изменения в зависимости от топологии и других функций).

- **spec.ipFamilies**

массив строк

Семейство IP-адресов (например, IPv4, IPv6), назначенных сервису. Это поле обычно назначается автоматически на основе конфигурации кластера и поля `ipFamilyPolicy`.

Если поле указано вручную и запрашиваемое семейство доступно в кластере, а `ipFamilyPolicy` это позволяет, оно будет использовано; в противном случае создание сервиса завершится ошибкой. Это поле можно изменять условно: допускается добавление или удаление вторичного IP-семейства, но нельзя изменить основное семейство IP-адресов сервиса.

Допустимые значения — `IPv4` и `IPv6`. Это поле применяется только к сервисам типов ClusterIP, NodePort и LoadBalancer.

Это поле будет удалено при обновлении сервиса до типа `ExternalName`.

Эти семейства должны соответствовать значениям поля `clusterIPs`, если оно указано. Поля `clusterIPs` и `ipFamilies` управляются полем `ipFamilyPolicy`. Поле может содержать максимум два значения.

- **Элемент массива**

строка

`IPFamily` представляет семейство IP-адресов (IPv4 или IPv6).

Этот тип используется для указания семейства IP-адресов, выраженного типом, например `service.spec.ipFamilies`.

- **spec.ipFamilyPolicy**

строка

Требуемая или запрашиваемая поддержка двух стеков для сервиса.

Если значение не указано, это поле будет установлено в `SingleStack`. Сервисы могут быть `SingleStack` (одно IP-семейство), `PreferDualStack` (два IP-семейства на кластерах с двойным стеком или одно IP-семейство на кластерах с одним стеком) или `RequireDualStack` (два семейства IP-адресов на кластерах с двойным стеком, в противном случае возникнет ошибка).

Поля `ipFamilies` и `clusterIPs` зависят от значения этого поля. Это поле будет очищено при обновлении сервиса до типа `ExternalName`.

- **spec.loadBalancerClass**

строка

Класс реализации балансировщика нагрузки, к которому принадлежит этот сервис. Если указано, значение этого поля должно быть идентификатором в стиле лейбла с необязательным префиксом, например, `internal-vip` или `example.com/internal-vip`. Имена без префикса зарезервированы для конечных пользователей.

Это поле может быть установлено только тогда, когда тип сервиса — `LoadBalancer`.

Если тип не указан, используется реализация балансировщика нагрузки по умолчанию, которая обычно осуществляется через интеграцию с облачным провайдером, но должна применяться для любой реализации по умолчанию.

Если тип указан, предполагается, что реализация балансировщика нагрузки отслеживает сервисы с соответствующим классом. Любая реализация балансировщика нагрузки по умолчанию (например, облачные провайдеры) должна игнорировать сервисы, устанавливающие это поле. Это поле может быть установлено только при создании или обновлении сервиса до типа `LoadBalancer`.

После установки его нельзя изменить. Это поле будет очищено, когда сервис будет обновлен до типа, отличного от `LoadBalancer`.

- **spec.loadBalancerIP**
строка

Применяется только для типа сервиса `LoadBalancer`. Эта функция зависит от того, поддерживает ли облачный провайдер возможность указания `loadBalancerIP` при создании балансировщика нагрузки.

Это поле будет игнорироваться, если облачный провайдер не поддерживает данную функцию.

- **spec.loadBalancerSourceRanges**
массив строк

Если указано и поддерживается платформой, это ограничит трафик через облачный балансировщик нагрузки только для указанных клиентских IP-адресов.

Это поле будет игнорироваться, если облачный провайдер не поддерживает данную функцию. Подробнее.

- **spec.ports**
массив объектов

Список портов, которые открыты этим сервисом. Подробнее.

`ServicePort` содержит информацию о порте сервиса.

- **spec.ports.appProtocol**
строка

Протокол приложения для данного порта. Служит подсказкой для реализаций, позволяя предложить более расширенное поведение для поддерживаемых протоколов. Это поле следует стандартному синтаксису лейблов Kubernetes. Допустимые значения:

- Именования протоколов без префиксов — зарезервированы для стандартных имён служб IANA, в соответствии с RFC-6335.
- Именования с префиксом, определенные Kubernetes:
 - `kubernetes.io/h2c` — HTTP/2 с предварительным знанием в открытом виде, как описано в RFC;
 - `kubernetes.io/ws` — WebSocket в открытом виде, как описано в RFC;
 - `kubernetes.io/wss` — WebSocket через TLS, как описано в RFC.
- Другие протоколы должны использовать имена с определенными реализациями, например с такими как `myscompany.com/my-custom-protocol`.

- **spec.ports.name**
строка

Имя этого порта внутри службы соответствует стандарту `DNS_LABEL`.

Все порты внутри `ServiceSpec` должны иметь уникальные имена. При рассмотрении конечных точек для службы имя должно совпадать с полем `name` в `EndpointPort`. Имя не должно совпадать с полем `name`, если в этой службе определен только один `ServicePort`.

- **spec.ports.nodePort**
целочисленный

Порт на каждом узле, на котором эта служба открыта, когда тип равен `NodePort` или `LoadBalancer`. Обычно назначается системой. Если указано значение, оно должно быть в диапазоне и не должно использоваться, иначе операция завершится с ошибкой.

Если не указано, порт будет выделен, если этой службе это требуется.

Если это поле указано при создании службы, которая в этом не нуждается, создание завершится с ошибкой.

Это поле будет очищено при обновлении службы, например, при изменении типа с `NodePort` на `ClusterIP`. Подробнее.

- **spec.ports.port**
целочисленный

Обязательный параметр

Порт, который будет открыт этой службой.

- **spec.ports.protocol**

строка

IP-протокол для этого порта. Поддерживает TCP, UDP и SCTP.

По умолчанию: TCP

- **spec.ports.targetPort**

целочисленный

Номер порта для доступа к подам, на которые нацелен сервис.

Это поле игнорируется для сервисов с clusterIP=None и должно быть установлено равным полю port. Подробнее.

- **spec.publishNotReadyAddresses**

булевый

Указывает, что любые агенты, занимающиеся обработкой конечных точек данного сервиса, должны игнорировать статусы готовности или неготовности. Основное применение этого поля заключается в том, чтобы сервисы StatefulSet могли распространять записи SRV DNS для своих подов с целью обнаружения пиров.

Контроллеры Kubernetes, которые генерируют ресурсы Endpoints и EndpointSlice для сервисов, трактуют это как указание, что все конечные точки следует считать "готовыми", даже если поды не прошли пробы и не готовы.

- **spec.selector**

объект

Маршрутизирует трафик сервиса к подам, у которых ключи и значения лейблов соответствуют этому селектору. Если параметр не определен, предполагается, что у сервиса есть внешний процесс, управляющий его конечными точками, которые Kubernetes не будет изменять.

Применимо только к типам ClusterIP, NodePort и LoadBalancer.

Игнорируется, если тип — ExternalName. Подробнее.

- **spec.sessionAffinity**

строка

Используется для поддержания сессий с привязкой к клиенту. Должно быть указано значение либо ClientIP, либо None. Подробнее.

- **spec.sessionAffinityConfig**

объект

Содержит конфигурацию привязки сессий.

- **spec.sessionAffinityConfig.clientIP**

объект

Содержит конфигурацию привязки сессий на основе IP-адреса клиента.

- **spec.sessionAffinityConfig.clientIP.timeoutSeconds**

целочисленный

Указывает время удержания сессии для типа ClientIP в секундах.

Значение должно быть >0 && <=86400 (для 1 дня), если ServiceAffinity == ClientIP.

- **spec.trafficDistribution**

строка

Позволяет задать предпочтения по распределению трафика между конечными точками сервиса. Реализации могут учитывать это поле как рекомендацию, но не обязаны строго его соблюдать.

Если поле не задано, реализация применяет стратегию маршрутизации по умолчанию. Если установлено значение PreferClose, реализации должны приоритизировать конечные точки, которые топологически близки (например, в той же зоне).

- **spec.type**

строка

Определяет доступность сервиса.

По умолчанию используется ClusterIP. Допустимые варианты: ExternalName, ClusterIP, NodePort и LoadBalancer.

ClusterIP выделяет внутренний IP-адрес для балансировки нагрузки между конечными точками. Конечные точки определяются `selector` или, если он не указан, вручную с помощью объектов `Endpoints` или `EndpointSlice`.

Если значение `clusterIP` установлено на `None`, виртуальный IP не выделяется, и конечные точки публикуются как набор конечных точек, а не как виртуальный IP.

NodePort расширяет ClusterIP и выделяет порт на каждом узле, который маршрутизируется к тем же конечным точкам, что и `clusterIP`.

LoadBalancer расширяет NodePort и создает внешний балансировщик нагрузки (если поддерживается текущим облаком), который маршрутизируется к тем же конечным точкам, что и `clusterIP`.

ExternalName создает алиас этому сервису на указанный `externalName`. Некоторые другие поля не применяются к сервисам типа ExternalName.

Подробнее.

- **status**
объект

Определяет наблюдаемое состояние `ServiceWithHealthchecks`.

- **status.conditions**
массив объектов

Текущее состояние сервиса

Condition contains details for one aspect of the current state of this API Resource. This struct is intended for direct use as an array at the field path `.status.conditions`.

- **status.conditions.lastTransitionTime**
строка

Обязательный параметр

Последнее время изменения, когда условие перешло из одного состояния в другое. Это должно соответствовать времени изменения основного условия.

Если время неизвестно, можно использовать время изменения поля API.

- **status.conditions.message**
строка

Обязательный параметр

Понятное человеку сообщение, указывающее детали перехода.

Это может быть пустая строка.

Максимальная длина: 32768

- **status.conditions.observedGeneration**
целочисленный

Значение `.metadata.generation`, на основе которого было установлено состояние.

Например, если `.metadata.generation` равно 12, а `.status.conditions[x].observedGeneration` равно 9, то состояние устарело по отношению к текущему состоянию экземпляра.

Допустимые значения: `0 <= X`

- **status.conditions.reason**
строка

Обязательный параметр

Программный идентификатор, указывающий причину последнего перехода состояния. Производители конкретных типов состояний могут определить ожидаемые значения и их значения для этого поля, а также то, считаются ли эти

значения гарантированным API.

Значение должно быть строкой в `CamelCase`.

Это поле не может быть пустым.

Шаблон: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Длина: `1..1024`

- **status.conditions.status**

строка

Обязательный параметр

Состояние условия, одно из значений `True`, `False`, `Unknown`.

Допустимые значения: `True`, `False`, `Unknown`

- **status.conditions.type**

строка

Обязательный параметр

Тип условия в формате `CamelCase` или в виде `foo.example.com/CamelCase`.

Многие значения `.condition.type` последовательны для различных ресурсов, таких как `Available`, но поскольку произвольные условия могут быть полезными (см. `.node.status.conditions`), важна возможность разрешения конфликтов. Регулярное выражение, которому оно соответствует: `(dns1123SubdomainFmt/)?(qualifiedNameFmt)`

Шаблон: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/*)?([A-Za-z0-9]([A-Za-z0-9_]*[A-Za-z0-9])$`

Максимальная длина: `316`

- **status.endpointStatuses**

массив объектов

Текущее состояние службы.

- **status.endpointStatuses.failedProbes**

массив строк

Список проб, завершившихся неудачно во время последней проверки.

- **status.endpointStatuses.lastProbeTime**

строка

Время последней проверки пода (обновляется в зависимости от настройки `verboseStatus`).

- **status.endpointStatuses.lastTransitionTime**

строка

Время, когда состояние пробы последний раз перешло из одного статуса в другой.

- **status.endpointStatuses.nodeName**

строка

Узел, на котором запущен под.

- **status.endpointStatuses.podName**

строка

Обязательный параметр

Имя проверяемого пода.

- **status.endpointStatuses.probesSuccessful**

булевый

Указывает, были ли успешными все пользовательские пробы `ServiceWithHealthchecks`.

- **status.endpointStatuses.ready**

булевый

Указывает, прошел ли под стандартные проверки readiness Kubernetes.

- **status.healthcheckCondition**

объект

Сводка результатов проверок (health checks).

- **status.healthcheckCondition.endpoints**

целочисленный

Общее количество эндпоинтов (подов), для которых выполняются проверки.

- **status.healthcheckCondition.observedGeneration**

целочисленный

Последнее поколение (generation) ресурса, которое наблюдал контроллер.

- **status.healthcheckCondition.readyEndpoints**

целочисленный

Количество эндпоинтов (подов), успешно прошедших проверки.

- **status.loadBalancer**

объект

Содержит текущее состояние балансировщика нагрузки, если он присутствует.

- **status.loadBalancer.ingress**

массив объектов

Список, содержащий точки входа для балансировщика нагрузки. Трафик, предназначенный для сервиса, должен направляться на эти точки входа.

Представляет состояние точки входа балансировщика нагрузки: трафик, предназначенный для сервиса, должен направляться на эту точку входа.

- **status.loadBalancer.ingress.hostname**

строка

Устанавливается для точек входа балансировщика нагрузки, которые основаны на DNS (обычно балансировщики нагрузки AWS).

- **status.loadBalancer.ingress.ip**

строка

Устанавливается для точек входа балансировщика нагрузки, которые основаны на IP-адресах (обычно балансировщики нагрузки GCE или OpenStack).

- **status.loadBalancer.ingress.ipMode**

строка

Указывает, как ведет себя IP-адрес балансировщика нагрузки и может быть указан только при указании поля `ip`. Установка значения `VIP` означает, что трафик доставляется на узел с назначением, установленным на IP-адрес и порт балансировщика нагрузки. Установка значения `Proxy` указывает, что трафик доставляется на узел или под с назначением, установленным на IP-адрес узла и порт узла или IP-адрес пода и порт.

Реализации сервиса могут использовать эту информацию для настройки маршрутизации трафика.

- **status.loadBalancer.ingress.ports**

массив объектов

Список записей портов сервиса.

Если используется, каждый порт, определенный в сервисе, должен иметь соответствующую запись в этом списке.

- **status.loadBalancer.ingress.ports.error**

строка

Запись проблемы с портом сервиса.

Формат ошибки должен соответствовать следующим правилам:

- Встроенные значения ошибок должны быть указаны в этом файле

- и использовать имена в формате CamelCase.
- Специфические для облачного провайдера значения ошибок должны иметь имена, соответствующие формату `foo.example.com/CamelCase`.

Регулярное выражение, с которым это соответствует:
(dns1123SubdomainFmt/)?(qualifiedNameFmt)

Шаблон: `^([a-z0-9]([a-z0-9]*[a-z0-9])?(\.[a-z0-9]([a-z0-9]*[a-z0-9])?)*\/)?((([A-Za-z0-9]([A-Za-z0-9_]*)?[A-Za-z0-9])$`

Максимальная длина: 316

- **status.loadBalancer.ingress.ports.port**
целочисленный

Обязательный параметр

Номер порта сервиса, статус которого указан.

- **status.loadBalancer.ingress.ports.protocol**
строка

Обязательный параметр

Протокол порта сервиса, статус которого здесь записан.
Поддерживаемые значения: TCP, UDP, SCTP.

По умолчанию: TCP

Примеры

i

Чтобы создаваемые балансировщики ServiceWithHealthchecks работали, необходимо выполнение следующих условий:

- В сетевой политике пользовательского проекта, в котором будет создаваться ServiceWithHealthchecks, должно присутствовать правило, разрешающее входящий трафик из всех подов неймспейса `d8-service-with-healthchecks`:

```
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: d8-service-with-healthchecks
```

Подробнее о сетевых политиках — в разделе «Настройка сетевых политик».

- Кластерная роль, которая используется в ClusterRoleBinding и RoleBinding при назначении прав пользователям и сервисным аккаунтам, для ресурса ServiceWithHealthchecks должна быть расширена следующими правилами:

- get
- list
- watch
- create
- update
- patch
- delete.

Подробнее — в разделе «Выдача прав пользователям и сервисным аккаунтам».



После включения модуля не происходит автоматическая замена имеющихся ресурсов типа Service на ServiceWithHealthcheck. Для замены имеющихся сервисов на использование ServiceWithHealthcheck выполните следующие действия:

- Создайте ресурсы ServiceWithHealthcheck с такими же именами и параметрами, как существующие ресурсы Service, которые нужно заменить. При создании ServiceWithHealthcheck укажите обязательные параметры `healthchecks`.
- Удалите ресурсы Service, которые требуется заменить ServiceWithHealthcheck.

Размещение двух независимых балансировщиков на одной виртуальной машине

На виртуальной машине с операционной системой Linux работают два приложения — HTTP-сервер (TCP 8080) и SMTP-сервер (TCP 2525). Необходимо настроить два отдельных балансировщика для этих сервисов — веб-балансировщик и SMTP-балансировщик.

Создание виртуальной машины

Создайте виртуальную машину `my-vm` основываясь на примерах из документации DVP.

В примере манифеста ниже добавлен лейбл `vm: my-vm` для дальнейшей идентификации в балансировщиках.

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: my-ns
  labels:
    vm: my-vm
spec:
  virtualMachineClassName: generic
  cpu:
    cores: 1
  memory:
    size: 1Gi
  provisioning:
    type: UserData
    userData: |
      #cloud-config
      ssh_pwauth: True
      users:
        - name: cloud
          passwd:
            '$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo1RquzS0R6GgdIDlvS.kbUwDablgZKZcTP/'
          shell: /bin/bash
          sudo: ALL=(ALL) NOPASSWD:ALL
          lock_passwd: False
      blockDeviceRefs:
        - kind: VirtualDisk
          name: linux-disk
```

Манифесты балансировщиков для веб-сервиса и SMTP

Пример манифеста веб-балансировщика:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: web
  namespace: my-ns
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
  selector:
    vm: my-vm
  healthcheck:
    probes:
      - mode: HTTP
        http:
          targetPort: 8080
          method: GET
          path: /healthz

```

Пример манифеста SMTP-балансировщика:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: smtp
  namespace: my-ns
spec:
  ports:
    - port: 25
      protocol: TCP
      targetPort: 2525
  selector:
    vm: my-vm
  healthcheck:
    probes:
      - mode: TCP
        tcp:
          targetPort: 2525

```

Балансировщики для работы с PostgreSQL-кластером

Создание StatefulSet для PostgreSQL

Для корректной работы `StatefulSet` потребуется создать стандартный сервис (Service) для формирования DNS-имени отдельных подов. Этот сервис не будет использоваться для прямого доступа к базе данных.

```

apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432

```

Пример манифеста `StatefulSet`:


```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  name: my-ns
spec:
  serviceName: postgres
  replicas: 3
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:13
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: postgres
            - name: POSTGRES_PASSWORD
              value: example

```

Конфигурация балансировщиков ServiceWithHealthchecks

Создайте Secret для хранения учетных данных для доступа проб к базе данных:

```

d8 k -n my-ns create secret generic cred-secret --from-literal=user=postgres --from-literal=password=example cred-secret

```

Пример манифеста балансировщика для чтения:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: postgres-read
spec:
  ports:
    - port: 5432
      protocol: TCP
      targetPort: 5432
  selector:
    app: postgres
  healthcheck:
    probes:
      - mode: PostgreSQL
        postgresSQL:
          targetPort: 5432
          dbName: postgres
          authSecretName: cred-secret
          query: "SELECT 1"

```

Пример манифеста балансировщика для записи:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: postgres-write
spec:
  ports:
    - port: 5432
      protocol: TCP
      targetPort: 5432
  selector:
    app: postgres
  healthcheck:
    probes:
      - mode: PostgreSQL
        postgresSQL:
          targetPort: 5432
          dbName: postgres
          authSecretName: cred-secret
          query: "SELECT NOT pg_is_in_recovery()"

```

Передача аннотаций и лейблов в дочерний Service

Аннотации и лейблы ServiceWithHealthchecks копируются в Service, который создает модуль. Это необходимо контроллерам, читающим параметры балансировщика только из Service, например MetalLB:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: postgres-read
  annotations:
    network.deckhouse.io/load-balancer-ips: 192.168.217.217
    network.deckhouse.io/load-balancer-shared-ip-key: key-to-share
spec:
  type: LoadBalancer
  ports:
    - port: 5432
      protocol: TCP
      targetPort: 5432
  selector:
    app: postgres
  healthcheck:
    probes:
      - mode: TCP
        tcp:
          targetPort: 5432

```

Список скопированных при последней синхронизации ключей хранится в аннотациях `network.deckhouse.io/propagated-annotations` и `network.deckhouse.io/propagated-labels` дочернего Service. Благодаря этим спискам ключ, удаленный из ServiceWithHealthchecks, удаляется и из Service, а аннотации и лейблы, предоставленные другими контроллерами (например, `metalb.universe.tf/ip-allocated-from-pool`), остаются нетронутыми.

Модуль terraform-manager

Модуль отвечает за отслеживание и синхронизацию состояния базовой инфраструктуры и постоянных узлов в облачной среде. Реализация основана на Terraform и применяется в Deckhouse Kubernetes Platform для взаимодействия с поддерживаемыми облачными провайдерами.

- Модуль состоит из двух частей:
 - `terraform-auto-converger` — проверяет состояние Terraform и применяет неструктивные изменения;

- `terraform-state-exporter` — проверяет состояние Terraform и экспортирует метрики кластера.
- Модуль включен по умолчанию, если в кластере есть секреты:
 - `kube-system/d8-provider-cluster-configuration`;
 - `d8-system/d8-cluster-terraform-state`.

Настройки

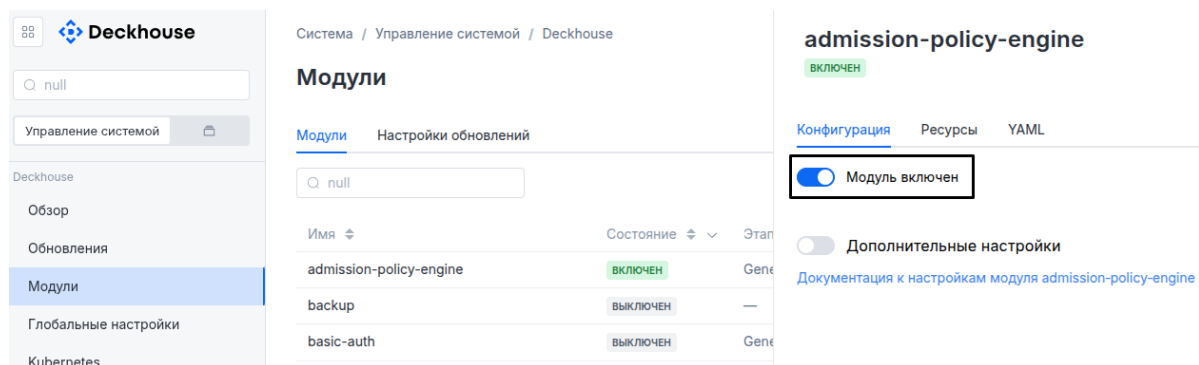
В модуле реализовано 10 алертов.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `terraform-manager`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `terraform-manager`:

```
d8 system module enable terraform-manager
```

- **С помощью ModuleConfig terraform-manager.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `terraform-manager` (создайте его, при необходимости).

Пример манифеста для включения модуля `terraform-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: terraform-manager
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `terraform-manager`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию settings**. Сохраните изменения.

Пример:

Конфигурация

Ресурсы

YAML



Модуль включен



Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация

YAML

Вы также можете отредактировать объект ModuleConfig `terraform-manager` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `terraform-manager`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `terraform-manager` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc terraform-manager
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `terraform-manager`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `terraform-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: terraform-manager
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Параметры

Версия схемы: 1

- **settings**
объект
 - **settings.autoConvergerEnabled**
булевый

Включает периодическую проверку состояния Terraform'а и применение неструктивных изменений.

Периодичность проверки задается параметром `autoConvergerPeriod`.

По умолчанию: `true`

- **settings.autoConvergerPeriod**
строка

Промежуток времени, через который проверяется состояние Terraform'a.

Задается в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

По умолчанию: `1h`

Шаблон: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.nodeSelector**
объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations**
массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Мониторинг SLA кластера

i

Веб-интерфейсы, связанные с модулем: `status`, `upmeter`

Модуль тестирует доступность платформы и состояние компонентов кластера в реальном времени, выводит информацию в виде соответствующих дашбордов.

Возможности модуля:

- для всех основных компонентов кластера разработаны сценарии постоянного тестирования;
- результаты тестирования сохраняются в виде метрик;
- отображает дашборд, показывающий работоспособность компонентов;
- данные могут быть экспортированы в любую внешнюю Prometheus-совместимую систему мониторинга.

Состав модуля:

- **agent** — работает на master-узлах и делает пробы доступности, отправляет результаты на сервер.
- **upmeter** — собирает результаты и поддерживает API-сервер для их извлечения.
- **front:**
 - **status** — показывает уровень доступности за последние 10 минут (требуется авторизация, но ее можно отключить);
 - **webui** — показывает дашборд со статистикой по пробам и группам доступности (требуется авторизация).
- **smoke-mini** — поддерживает постоянное *smoke-тестирование* с помощью StatefulSet.

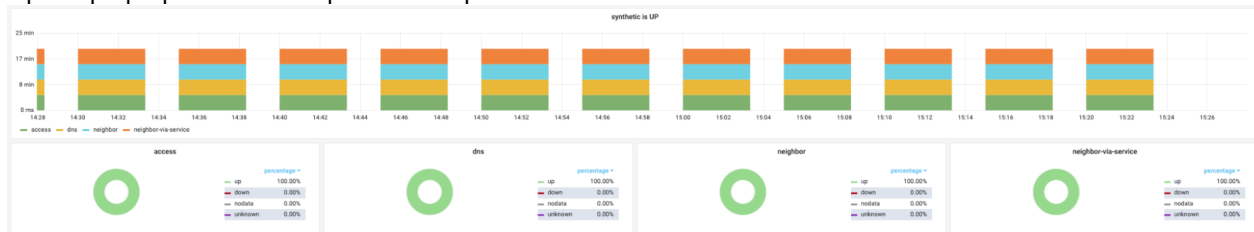
Модуль отправляет около 100 показаний метрик каждые 5 минут. Это значение зависит от количества включенных модулей Deckhouse Kubernetes Platform.

Интерфейс

Пример веб-интерфейса:



Пример графиков по метрикам из upmeter в Grafana:



Настройки

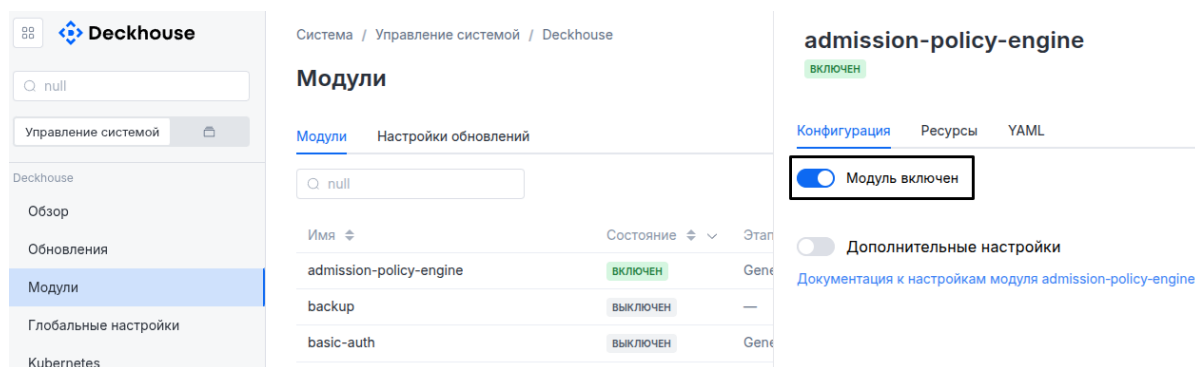
В модуле реализовано 15 алертов.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `upmeter`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable upmeter` для включения модуля, или `d8 system module disable upmeter` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `upmeter`:

```
d8 system module enable upmeter
```

- **С помощью ModuleConfig upmeter.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig upmeter` (создайте его, при необходимости).

Пример манифеста для включения модуля `upmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  enabled: true
```

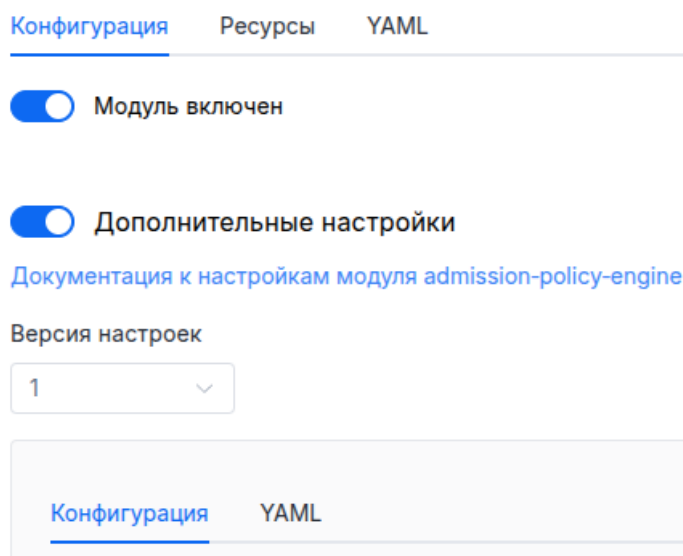
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `upmeter`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект `ModuleConfig upmeter` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `upmeter`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig upmeter` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc upmeter
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig upmeter`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig upmeter`:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  version: 3
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Описание конверсии отсутствует.

Преобразования, выполняемые конверсией (синтаксис jq)...

- `del(.auth.status.password) | if .auth.status == {} then del(.auth.status) end | if .auth == {} then del(.auth) end`
- `del(.auth.webui.password) | if .auth.webui == {} then del(.auth.webui) end | if .auth == {} then del(.auth) end`

- Обновление из версии **2** в **3**:

Описание конверсии отсутствует.

Преобразования, выполняемые конверсией (синтаксис jq)...

- `del(.smokeMini.ingressClass) | if .smokeMini == {} then del(.smokeMini) end`

Параметры

Версия схемы: 3

- **settings**

объект

- **settings.auth**

объект

Обязательный параметр

Доступ к веб-интерфейсу.

- **settings.auth.status**

объект

Обязательный параметр

Настройки доступа к веб-интерфейсу.

- **settings.auth.status.allowedUserEmails**

массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.status.allowedUserGroups**

массив строк

Группы доступа к просмотру интерфейса приложения.

- Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.

- **Внимание!** При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее

поле в настройках DexProvider.

- **settings.auth.status.externalAuthentication**
объект

Параметры для подключения внешней аутентификации (используется механизм Ingress NGINX external-auth, работающий на основе модуля NGINX auth_request).



Внешняя аутентификация включается автоматически, если включен модуль user-authn.

- **settings.auth.status.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

- **settings.auth.status.externalAuthentication.authURL**
строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

- **settings.auth.status.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **Элемент массива**
строка

Шаблон: `^(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])|([0-9]|[12][0-9]|3[0-2]))?|([0-9A-Fa-f]{0,4}:){2,7}[0-9A-Fa-f]{0,4}|([0-9]|1[0-9]|2[0-8]))?)$`

- **settings.auth.webui**
объект

Обязательный параметр

Настройки доступа к веб-интерфейсу.

- **settings.auth.webui.allowedUserEmails**
массив строк

Массив адресов электронной почты, пользователям которых разрешен доступ к веб-интерфейсу.

Используется, если включен модуль `user-authn` или задан параметр `externalAuthentication`.

- **settings.auth.webui.allowedUserGroups**
массив строк

Группы доступа к просмотру интерфейса приложения.

- Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.

- **Внимание!** При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках DexProvider.

- **settings.auth.webui.externalAuthentication**
объект

Параметры для подключения внешней аутентификации (используется

механизм Ingress NGINX external-auth, работающий на основе модуля NGINX auth_request.

i Внешняя аутентификация включается автоматически, если включен модуль user-authn.

■ **settings.auth.webui.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

■ **settings.auth.webui.externalAuthentication.authURL**
строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

■ **settings.auth.webui.password**
строка

Password for http authorization of the `admin` user. It is generated automatically, but you can change it.

This parameter is used if the `externalAuthentication` is not enabled.

■ **settings.auth.webui.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

■ **Элемент массива**
строка

Шаблон: `^(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\/([0-9]{1,2}|12[0-9]|13[0-2]))?|([0-9A-Fa-f]{0,4}:){2,7}[0-9A-Fa-f]{0,4}\/([0-9]{1,2}|1[0-9]|10[0-9]|11[0-9]|12[0-8]))?)$`

○ **settings.disabledProbes**
массив строк

Массив строк из названий групп или определенных проб из группы. Названия можно посмотреть в веб-интерфейсе.

Пример:

```
disabledProbes:
- "synthetic/api" # Отключить отдельную пробу.
- "synthetic/"    # Отключить группу проб.
- control-plane  # Или без /.
```

По умолчанию: `[]`

○ **settings.https**
объект

Тип сертификата, используемого для веб-приложений webui и status.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.https.certManager**

объект

■ **settings.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для webui/status.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

■ **settings.https.customCertificate**

объект

■ **settings.https.customCertificate.secretName**

строка

Имя Secret'a в пространстве имен `d8-system`, который будет использоваться для webui/status.

Secret должен быть в формате TLS.

По умолчанию: `false`

■ **settings.https.mode**

строка

Режим работы HTTPS:

- `Disabled` — webui/status будут работать только по HTTP;
- `CertManager` — webui/status будут работать по HTTPS и заказывать сертификат с помощью ClusterIssuer, заданного в параметре `certManager.clusterIssuerName`;
- `CustomCertificate` — webui/status будут работать по HTTPS, используя сертификат из namespace `d8-system`;
- `OnlyInURI` — webui/status будет работать по HTTP (подразумевая, что перед ними стоит внешний балансировщик, который терминирует HTTPS) и все ссылки в `user-authn` будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

○ **settings.ingressClass**

строка

Класс Ingress-контроллера, который используется для Upmeter.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

○ **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
nodeSelector:
  disktype: ssd
```

- **settings.smokeMini**
объект

Configuration of authentication for smoke-mini.

- **settings.smokeMini.auth**
объект

Обязательный параметр

Настройки доступа к веб-интерфейсу.

- **settings.smokeMini.auth.allowedUserGroups**
массив строк

Группы доступа к просмотру интерфейса приложения.

- Используется, если включен модуль `user-authn` или параметр `externalAuthentication`.
- **Внимание!** При использовании совместно с модулем `user-authn` необходимо также добавить разрешенные группы в соответствующее поле в настройках `DexProvider`.

- **settings.smokeMini.auth.externalAuthentication**
объект

Параметры для подключения внешней аутентификации. Используется механизм Ingress NGINX `external-auth`, работающей на основе модуля NGINX `auth_request`.

- **settings.smokeMini.auth.externalAuthentication.authSignInURL**
строка

URL, куда будет перенаправлен пользователь для прохождения аутентификации (если сервис аутентификации вернул код ответа HTTP, отличный от 200).

- **settings.smokeMini.auth.externalAuthentication.authURL**
строка

URL сервиса аутентификации. Если пользователь прошел аутентификацию, сервис должен возвращать код ответа HTTP 200.

- **settings.smokeMini.auth.password**
строка

Пароль для HTTP-авторизации для пользователя `admin` (генерируется автоматически, но можно менять)

Используется, если не включен параметр `externalAuthentication`.

- **settings.smokeMini.auth.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **Элемент массива**
строка

Шаблон: `^(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\/([0-9]|[12][0-9]|3[0-2]))?|([0-9A-Fa-f]{0,4}:){2,7}[0-9A-Fa-f]{0,4}\/([0-9]|[1-9][0-9]|1[01][0-9]|12[0-8]))?)$`

- **settings.smokeMini.https**
объект

Тип сертификата, используемого для smoke-mini.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

■ **settings.smokeMini.https.certManager**

объект

■ **settings.smokeMini.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для smoke-mini.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

■ **settings.smokeMini.https.customCertificate**

объект

■ **settings.smokeMini.https.customCertificate.secretName**

строка

Имя Secret'a в namespace `d8-system`, который будет использоваться для smoke-mini.

Secret должен быть в формате TLS.

По умолчанию: `false`

■ **settings.smokeMini.https.mode**

строка

Режим работы HTTPS:

- `Disabled` — smoke-mini будут работать только по HTTP;
- `CertManager` — smoke-mini будут работать по HTTPS и заказывать сертификат с помощью ClusterIssuer, заданного в параметре `certManager.clusterIssuerName`.
- `CustomCertificate` — smoke-mini будут работать по HTTPS, используя сертификат из namespace `d8-system`.
- `OnlyInURI` — smoke-mini будет работать по HTTP (подразумевая, что перед ними стоит внешний балансировщик, который терминирует HTTPS) и все ссылки в user-authn будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

■ **settings.smokeMini.storageClass**

строка or булевый

StorageClass для использования при проверке работоспособности дисков.

Если значение не указано, то будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Если указать `false`, будет принудительно использоваться `emptyDir`.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск будет перезаказан, и все данные удалятся.

Внимание. При переносе Upmeter с локальным хранилищем на другие узлы, поды зависнут в состоянии Pending. В этом случае будет необходимо сохранить базу данных Upmeter, удалить старые PVC и перезапустить под вручную. Под локальным хранилищем подразумевается StorageClass, связанный не с сетевым хранилищем, а с локальным томом на узле (например, StorageClass, создаваемый модулем local-path-provisioner).

По умолчанию: `false`

Допустимые значения: `false`

Примеры:

```
storageClass: false
```

```
storageClass: default
```

- **settings.smokeMiniDisabled**
булевый

Выключение smokeMini. Также отключает группу “synthetic”.

По умолчанию: `false`

- **settings.statusPageAuthDisabled**
булевый

Выключение авторизации для status-домена.

По умолчанию: `false`

- **settings.storageClass**
строка or булевый

Имя StorageClass для хранения данных Upmeter (PVC).

Если значение не указано, используется StorageClass существующей PVC. Если PVC нет, будет использоваться StorageClass, согласно настройке глобального параметра `storageClass`.

Настройка глобального параметра `storageClass` учитывается только при включении модуля. Изменение глобального параметра `storageClass` при включенном модуле не приведет к перезаказу диска.

Если указать `false`, будет принудительно использоваться `emptyDir`.

Внимание. Если указать значение, отличное от текущего (используемого в существующей PVC), диск будет перезаказан, и все данные удалятся.

Внимание. При переносе Upmeter с локальным хранилищем на другие узлы, поды зависнут в состоянии Pending. В этом случае будет необходимо сохранить базу данных Upmeter, удалить старые PVC и перезапустить под вручную. Под локальным хранилищем подразумевается StorageClass, связанный не с сетевым хранилищем, а с локальным томом на узле (например, StorageClass, создаваемый модулем local-path-provisioner).

Допустимые значения: `false`

Примеры:

```
storageClass: false
```

```
storageClass: default
```

- **settings.tolerations**
массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**
строка
- **settings.tolerations.key**
строка
- **settings.tolerations.operator**
строка
- **settings.tolerations.tolerationSeconds**
целочисленный
- **settings.tolerations.value**
строка

Аутентификация

По умолчанию используется модуль `user-authn`. Также можно настроить аутентификацию через `externalAuthentication` (см. ниже). Если эти варианты отключены, то модуль включит `basic auth` со сгенерированным паролем.

Посмотреть сгенерированный пароль можно командой:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.webui.password'
```

Чтобы сгенерировать новый пароль, нужно удалить Secret:

```
d8 k -n d8-upmeter delete secret/basic-auth-webui
```

Посмотреть сгенерированный пароль для страницы статуса можно командой:

```
d8 k -n d8-system exec svc/deckhouse-leader -C deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.status.password'
```

Чтобы сгенерировать новый пароль для страницы статуса, нужно удалить секрет:

```
d8 k -n d8-upmeter delete secret/basic-auth-status
```

i

Внимание. Параметры `auth.status.password` и `auth.webui.password` больше не поддерживаются.

Кастомные ресурсы

i

Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

Downtime

Scope: Cluster

Version: v1alpha1

Описание интервалов недоступности системы.

Используется для корректировки уровня SLA.

- **spec**

массив объектов

Обязательный параметр

- **spec.affected**

массив строк

Список групп проб модуля, которые были недоступны.

- **spec.description**

строка

Подробное описание.

- **spec.endDate**

строка

Время окончания (в формате Unix time или RFC3339, например `2020-09-07T17:24:55Z`)

- **spec.startDate**

строка

Время начала (в формате Unix time или RFC3339, например `2020-09-07T17:24:55Z`)

- **spec.type**

строка

Тип.

Допустимые значения: `Accident`, `Maintenance`, `InfrastructureMaintenance`, `InfrastructureAccident`

UpmeterHookProbe

Scope: Cluster

Version: v1

Ресурс для пробы работоспособности декхауса.

- **spec**

объект

Обязательный параметр

- **spec.inited**

строка

Обязательный параметр

Начальное значение, которое устанавливает upmeter-агент.

- **spec.mirror**

строка

Обязательный параметр

Значение, которое должно стать равным `initial` в результате работы хука.

UpmeterRemoteWrite

Scope: Cluster

- `v1`

- `v1alpha1`

Конфигурация отправки результатов проб модуля `upmeter` по протоколу `remote_write`.

- **spec**

объект

Обязательный параметр

- **spec.additionalLabels**

объект

Дополнительные лейблы для добавления ко всем метрикам.

- **spec.config**

объект

Обязательный параметр

Конфигурация протокола Prometheus `remote_write`.

- **spec.config.basicAuth**

объект

Параметры Basic-авторизации для endpoint'а.

- **spec.config.basicAuth.password**

строка

Обязательный параметр

Пароль для аутентификации.

- **spec.config.basicAuth.username**

строка

Обязательный параметр

Имя для аутентификации.

- **spec.config.bearerToken**

строка

Токен для аутентификации.

- **spec.config.headers**

объект

HTTP-заголовки, необходимые при отправке с каждым запросом.

- **spec.config.headers.X-Auth-Token**

строка

Заголовок `X-Auth-Token`, указывающий токен аутентификации.

- **spec.config.headers.X-Scope-OrgID**

строка

Заголовок `X-Scope-OrgID`, указывающий ID тенанта.

- **spec.config.tlsConfig**

объект

Параметры TLS.

- **spec.config.tlsConfig.ca**

строка

Сертификат ЦС (CA), используемый при проверке сертификата API-сервера.

По умолчанию:

- **spec.config.tlsConfig.insecureSkipVerify**

булевый

Отключить проверку сертификата.

По умолчанию: `false`

- **spec.config.url**

строка

Обязательный параметр

URL endpoint'а для отправки данных.

- **spec.intervalSeconds**

число

Временной интервал между точками (в секундах).

По умолчанию: `300`

Допустимые значения: 30, 300

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Конфигурация отправки результатов проб модуля `upmeter` по протоколу `remote_write`.

- **spec**

объект

Обязательный параметр

- **spec.additionalLabels**

объект

Дополнительные лейблы для добавления ко всем метрикам.

- **spec.config**

объект

Обязательный параметр

Конфигурация протокола Prometheus `remote_write`.

- **spec.config.basicAuth**

объект

Параметры Basic-авторизации для endpoint'a.

- **spec.config.basicAuth.password**

строка

Обязательный параметр

Пароль для аутентификации.

- **spec.config.basicAuth.username**

строка

Обязательный параметр

Имя для аутентификации.

- **spec.config.bearerToken**

строка

Токен для аутентификации.

- **spec.config.headers**

объект

HTTP-заголовки, необходимые при отправке с каждым запросом.

- **spec.config.headers.X-Auth-Token**

строка

Заголовок `X-Auth-Token`, указывающий токен аутентификации.

- **spec.config.headers.X-Scope-OrgID**

строка

Заголовок `X-Scope-OrgID`, указывающий ID тенанта.

- **spec.config.tlsConfig**

объект

Параметры TLS.

- **spec.config.tlsConfig.ca**

строка

Сертификат ЦС (CA), используемый при проверке сертификата API-сервера.

По умолчанию:

- **spec.config.tlsConfig.insecureSkipVerify**

булевый

Отключить проверку сертификата.

По умолчанию: `false`

- **spec.config.url**

строка

Обязательный параметр

URL endpoint'а для отправки данных.

- **spec.intervalSeconds**

число

Временной интервал между точками (в секундах).

По умолчанию: `300`

Допустимые значения: `30`, `300`

FAQ

Почему периодически не могут разместиться поды `upmeter-probe-controller-manager`, почему некоторые поды постоянно удаляются?

В модуле реализованы тесты доступности функционала различных контроллеров Kubernetes. Тесты выполняются путем создания и удаления временных подов.

Объекты `upmeter-probe-scheduler`, отвечают за реализацию теста функционала размещения подов на узлы. В рамках теста создается под, который размещается на узел. Затем этот под удаляется.

Объекты `upmeter-probe-controller-manager` отвечают за тестирование работоспособности `kube-controller-manager`.

В рамках теста создается StatefulSet, и проверяется что данный объект породил под (т.к. фактическое размещение пода не требуется и проверяется в рамках другого теста, то создается под который гарантированно не может разместиться, т.е. находится в состоянии Pending). Затем StatefulSet удаляется и выполняется проверка, что порожденный им под удалился.

Объекты `smoke-mini` реализуют тестирование сетевой связности между узлами. Для проверки размещаются пять StatefulSet с одной репликой. В рамках теста проверяется связность как между подами `smoke-mini`, так и сетевая связность с подами `upmeter-agent`, работающими на master-узлах.

Раз в минуту один из подов `smoke-mini` переносится на другой узел.

Примеры

Пример конфигурации `remote_write`

```
apiVersion: deckhouse.io/v1
kind: UpmeterRemoteWrite
metadata:
  labels:
    heritage: upmeter
    module: upmeter
    name: victoriametrics
spec:
  additionalLabels:
    cluster: cluster-name
    some: fun
  config:
    url: https://upmeter-victoriametrics.whatever/api/v1/write
    basicAuth:
      password: "Cdp#Cd.0xfZsx4*89SZ"
      username: upmeter
    intervalSeconds: 300
```

Модуль user-authn

i Веб-интерфейсы, связанные с модулем: kubeconfig

Модуль отвечает за единую систему аутентификации, интегрированную с Kubernetes и веб-интерфейсами, используемыми в других модулях, например, Grafana и Dashboard.

Модуль состоит из следующих компонентов:

- `dex` — федеративный OpenID Connect провайдер, поддерживающий работу со статическими пользователями и с возможностью подключения к различным внешним провайдерам аутентификации, например, SAML, GitLab, GitHub.
- `kubeconfig-generator` (он же `dex-k8s-authenticator`) — веб-приложение, генерирующее команды для настройки локального `kubectl` после аутентификации в Dex;
- `dex-authenticator` (он же `oauth2-proxy`) — приложение, которое получает запросы от компонента Ingress NGINX (через модуль `auth_request`) и авторизует их с помощью сервиса Dex.

Управление статическими пользователями осуществляется с помощью ресурсов `User` и `Group`:

В объекте `User` хранится информация о пользователе, включая email и зашифрованный хеш пароля (пароль в открытом виде не сохраняется); В объекте `Group` задаётся список пользователей, объединённых в группы для удобства управления правами доступа.

Поддерживаются следующие внешние провайдеры/протоколы аутентификации:

- GitHub;
- GitLab;
- BitBucket Cloud;
- Crowd;
- LDAP;
- OIDC.

Одновременно можно подключить более одного внешнего провайдера аутентификации.

Возможности интеграции

Базовая аутентификация в API Kubernetes

Базовая аутентификация в API Kubernetes на данный момент доступна только для провайдера Crowd (с включением параметра `enableBasicAuth`).

i К API Kubernetes можно подключаться и через другие поддерживаемые внешние провайдеры.

Интеграция с приложениями

Чтобы обеспечить аутентификацию в любом веб-приложении, работающем в Kubernetes, можно создать ресурс `DexAuthenticator` в пространстве имен (`Namespace`) приложения и добавить несколько аннотаций к ресурсу `Ingress`. Это позволит:

- ограничить список групп, которым разрешен доступ;
- ограничить список адресов, с которых разрешена аутентификация;
- интегрировать приложение в единую систему аутентификации, если приложение поддерживает OIDC. Для этого в Kubernetes создается ресурс `DexClient` в `Namespace` приложения. В том же `Namespace` создается секрет с данными для подключения в Dex по OIDC.

После такой интеграции можно:

- ограничить перечень групп, которым разрешено подключаться;
- указать перечень клиентов, OIDC-токенам которых можно доверять (`trustedPeers`).

Веб-интерфейс для генерации готовых kubeconfig-файлов

Модуль позволяет автоматически создавать конфигурацию для `kubectl` или других утилит Kubernetes.

Пользователь получит набор команд для настройки `kubectl` после авторизации в веб-интерфейсе генератора. Эти команды можно скопировать и вставить в консоль для использования `kubectl`. Механизм аутентификации для kubeconfig использует OIDC-токен. OIDC-сессия может

продлеваться автоматически, если использованный в Dex провайдер аутентификации поддерживает продление сессий. Для этого в kubeconfig указывается `refresh token`.

Дополнительно можно настроить несколько адресов `kube-apiserver` и сертификаты ЦС (CA) для каждого из них. Например, это может потребоваться, если доступ к кластеру Kubernetes осуществляется через VPN или прямое подключение.

Публикация API kubernetes через Ingress

Компонент kube-apiserver без дополнительных настроек доступен только во внутренней сети кластера. Этот модуль решает проблему простого и безопасного доступа к API Kubernetes извне кластера. При этом API-сервер публикуется на специальном домене (подробнее см. раздел о служебных доменах в документации).

При настройке можно указать:

- перечень сетевых адресов, с которых разрешено подключение;
- перечень групп, которым разрешен доступ к API-серверу;
- Ingress-контроллер, на котором производится аутентификация.

По умолчанию будет сгенерирован специальный сертификат ЦС (CA) и автоматически настроен генератор kubeconfig.

Расширения от Фланта

Модуль использует модифицированную версию Dex для поддержки:

- групп для статических учетных записей пользователей и провайдера Bitbucket Cloud (параметр `bitbucketCloud`);
- передачи параметра `group` клиентам;
- механизма `obsolete tokens`, который позволяет избежать состояния гонки при продлении токена OIDC-клиентом.
- SPNEGO (Kerberos) для LDAP-коннектора. При использовании этого механизма Dex принимает билеты по заголовку `Authorization: Negotiate`, валидирует их по keytab, и завершает вход без отображения формы ввода логина/пароля.

Отказоустойчивый режим

Модуль поддерживает режим высокой доступности `highAvailability`. При его включении аутентификаторы, отвечающие на `auth request`-запросы, развертываются с учетом требуемой избыточности для обеспечения непрерывной работы. В случае отказа любого из экземпляров аутентификаторов пользовательские аутентификационные сессии не прерываются.

Настройки

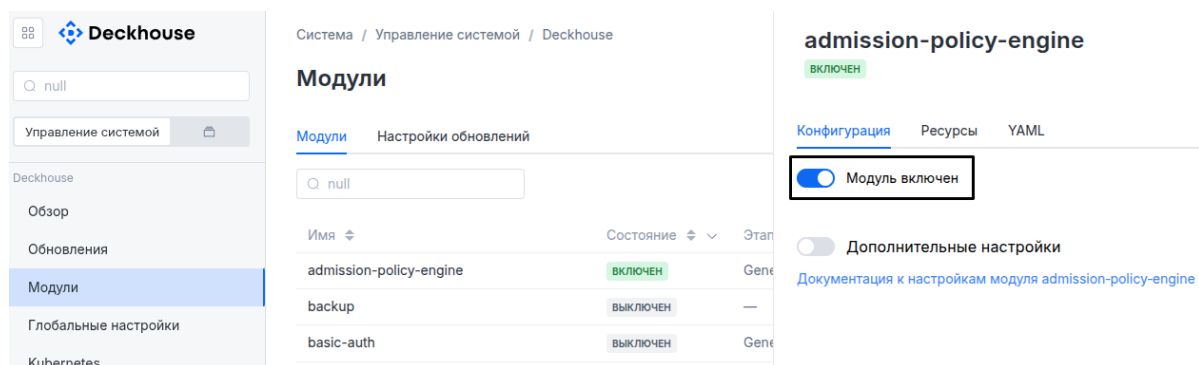
В модуле реализовано 3 алерта.

Модуль **включен** по умолчанию в наборе модулей `Default`. Модуль **выключен** по умолчанию в наборах модулей: `Managed`, `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authn`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `user-authn`:

```
d8 system module enable user-authn
```

- **С помощью ModuleConfig `user-authn`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `user-authn` (создайте его, при необходимости).

Пример манифеста для включения модуля `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  enabled: true
```

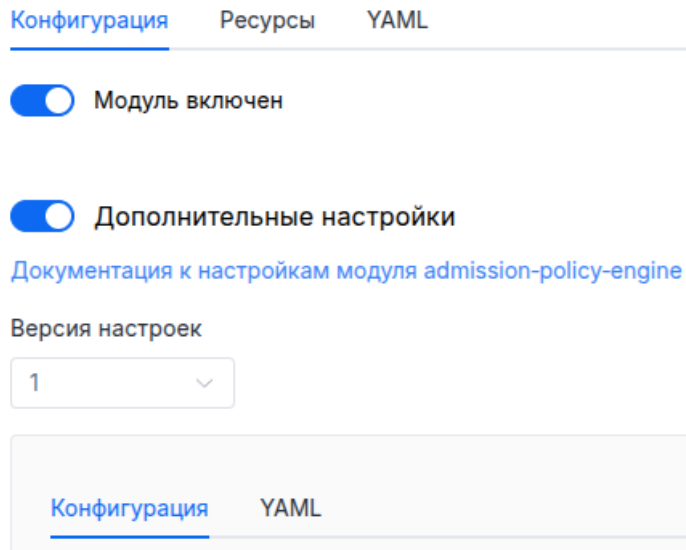
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authn`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект ModuleConfig `user-authn` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authn`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `user-authn` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc user-authn
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `user-authn`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `user-authn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Конверсии

Модуль настраивается с помощью ресурса ModuleConfig, схема которого содержит номер версии. При применении в кластере *старой* версии схемы ModuleConfig выполняются автоматические преобразования. Чтобы обновить версию схемы ModuleConfig вручную, необходимо **последовательно** для каждой версии выполнить следующие действия :

- Обновление из версии **1** в **2**:

Замените `publishAPI.enable` на `publishAPI.enabled`.

Параметры

Версия схемы: 2

- **settings**
объект
 - **settings.controlPlaneConfigurator**
объект

Параметры модуля `control-plane-manager` для автоматической настройки `kube-apiserver`.

- **settings.controlPlaneConfigurator.dexCAMode**
строка

Способ определения CA, который будет использован при настройке `kube-apiserver`:

- `Custom` — использовать CA, указанный в параметре `dexCustomCA`. Этот вариант уместен, например, если вы используете внешний HTTPS-балансировщик перед Ingress-контроллером и на этом балансировщике используется самоподписанный сертификат.
- `DoNotNeed` — CA не требуется (например, при использовании публичного TLS-провайдера Let's Encrypt или других).
- `FromIngressSecret` — извлечь CA или сам сертификат из Secret'a, который используется в Ingress-ресурсе. Если вы используете самоподписанные сертификаты в Ingress-ресурсах — это ваш вариант.

По умолчанию: `DoNotNeed`

Допустимые значения: `Custom`, `DoNotNeed`, `FromIngressSecret`

- **settings.controlPlaneConfigurator.dexCustomCA**
строка

CA, который будет использован в случае, если `dexCAMode` установлен в `Custom`.

- **settings.controlPlaneConfigurator.enabled**
булевый

Использовать ли `control-plane-manager` для настройки OIDC в `kube-apiserver`.

По умолчанию: `true`

- **settings.highAvailability**
булевый

Ручное управление режимом отказоустойчивости.

По умолчанию режим отказоустойчивости определяется автоматически. Подробнее про режим отказоустойчивости.

Примеры:

```
highAvailability: true
```

```
highAvailability: false
```

- **settings.https**
объект

Тип сертификата, используемого для Dex/kubeconfig-generator.

При использовании этого параметра полностью переопределяются глобальные настройки `global.modules.https`.

Примеры:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**

объект

- **settings.https.certManager.clusterIssuerName**

строка

ClusterIssuer, используемый для Dex/kubeconfig-generator.

Доступны `letsencrypt`, `letsencrypt-staging`, `selfsigned`, но вы можете определить свои.

По умолчанию: `letsencrypt`

- **settings.https.customCertificate**

объект

- **settings.https.customCertificate.secretName**

строка

Имя Secret'a в namespace `d8-system`, который будет использоваться для Dex/kubeconfig-generator.

Secret должен быть в формате `kubernetes.io/tls`.

По умолчанию: `false`

- **settings.https.mode**

строка

Режим работы HTTPS:

- `CertManager` — Dex/kubeconfig-generator будет работать по HTTPS и заказывать сертификат с помощью ClusterIssuer, заданного в параметре `certManager.clusterIssuerName`;
- `CustomCertificate` — Dex/kubeconfig-generator будет работать по HTTPS, используя сертификат из namespace `d8-system`;
- `Disabled` — Dex/kubeconfig-generator будет работать только по HTTP;
- `OnlyInURI` — Dex/kubeconfig-generator будет работать по HTTP

(подразумевая, что перед ним стоит внешний балансировщик, который терминирует HTTPS-трафик) и все ссылки в user-authn будут генерироваться с HTTPS-схемой. Балансировщик должен обеспечивать перенаправление с HTTP на HTTPS.

По умолчанию: `Disabled`

Допустимые значения: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.idTokenTTL**
строка

Время жизни ID-токена.

Задается в виде строки с указанием часов, минут и секунд: `30m`, `20s`, `2h30m10s`, `24h`.

Задаёт минимально возможное время жизни пользовательской сессии (параметр `keepUsersLoggedInFor` ресурса `DexAuthenticator` и параметры вида `auth.sessionTTL`, например, в модуле `console`).

Если время жизни сессии меньше или равно `idTokenTTL`, фактическое время до выхода пользователя из системы составит `idTokenTTL + 1s`.

Чтобы пользователь выходил из системы раньше значения `idTokenTTL` по умолчанию, уменьшите также значение этого параметра.

По умолчанию: `10m`

Шаблон: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.ingressClass**
строка

Класс Ingress-контроллера, который используется для Dex/kubeconfig-generator.

Опциональный параметр, по умолчанию используется глобальное значение `modules.ingressClass`.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.kubeconfigGenerator**
массив объектов

Массив, в котором указываются дополнительные способы доступа к API-серверу.

Параметр может быть полезен в случае, если вы хотите предоставить доступ к API-серверу не через Ingress, а другими способами (например, с bastion-хоста или через OpenVPN).

- **settings.kubeconfigGenerator.description**
строка

Текстовое описание, содержащее информацию о том, чем этот метод аутентификации отличается от других.

- **settings.kubeconfigGenerator.id**
строка

Обязательный параметр

Имя способа доступа к API-серверу (без пробелов, маленькими буквами).

Шаблон: `^[@.\:0-9a-z._-]+$`

- **settings.kubeconfigGenerator.masterCA**
строка

CA для доступа к API-серверу.

Если данный параметр не указать, то будет автоматически использован Kubernetes CA.

При публикации API-сервера через HTTP-прокси, который терминирует HTTPS-трафик, рекомендуется использовать самоподписанный сертификат, который нужно указать в параметре `masterCA`.

- **settings.kubeconfigGenerator.masterURI**
строка

Обязательный параметр

Если планируется использовать TCP-прокси, для адреса TCP-прокси должен быть сконфигурирован сертификат на стороне API-сервера. Например, в случае, если API-сервер доступен по трем адресам (192.168.0.10, 192.168.0.11 и 192.168.0.12), а ходить к API-серверу клиент будет через TCP-балансер (например, 192.168.0.15), необходимо регенерировать сертификаты для API-серверов:

- отредактировать `kubeadm-config: d8 k -n kube-system edit configmap kubeadm-config`, добавив в `.apiServer.certSANs` адрес 192.168.0.15;
- сохранить получившийся конфиг: `kubeadm config view > kubeadmconf.yaml`;
- удалить старые сертификаты API-сервера: `mv /etc/kubernetes/pki/apiserver.* /tmp/`;
- перевыпустить новые сертификаты: `kubeadm init phase certs apiserver - -config=kubeadmconf.yaml`;
- перезапустить контейнер с API-сервером: `docker ps -a | grep 'kube-apiserver' | grep -v pause | awk '{print $1}' | xargs docker restart`;
- повторить данное действие для всех master-узлов.

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.passwordPolicy**

объект

По умолчанию:

- **settings.passwordPolicy.complexityLevel**

строка

Уровень сложности пароля.

В зависимости от уровня сложности к паролю предъявляются следующие минимальные требования:

- `None` — 1 символ.
- `Low` — 8 символов.
- `Fair` — 8 символов, 1 заглавная буква (A-Z), 1 строчная буква (a-z), 1 цифра.
- `Good` — 8 символов, 1 заглавная буква (A-Z), 1 строчная буква (a-z), 1 цифра, 1 специальный символ (!@#\$%^&*).
- `Excellent` — 8 символов, 1 заглавная буква (A-Z), 1 строчная буква (a-z), 1 цифра, 1 специальный символ (!@#\$%^&*), не более 2 одинаковых символов подряд.

По умолчанию: `Fair`

Допустимые значения: `None`, `Low`, `Fair`, `Good`, `Excellent`

- **settings.passwordPolicy.lockout**

объект

Настройки временной блокировки пользователя при последовательных неуспешных попытках входа.

- **settings.passwordPolicy.lockout.lockDuration**

строка

Обязательный параметр

Длительность временной блокировки. Задается в секундах, минутах, часах или днях: `10s`, `5m`, `1h`, `3d`.

Шаблон: `^(?:(?:[1-9]|[1-9][0-9]+)d)?(?:(?:[1-9]|[1-9][0-9]+)h)?(?:(?:[1-9]|[1-9][0-9]+)m)?(?:(?:[1-9]|[1-9][0-9]+)s)?$`

- **settings.passwordPolicy.lockout.maxAttempts**

целочисленный

Обязательный параметр

Количество последовательных неуспешных попыток входа, после которых

пользователь будет временно заблокирован.

- **settings.passwordPolicy.passwordHistoryLimit**

целочисленный

Количество предыдущих паролей пользователя, которые сохраняются в истории для предотвращения их повторного использования.

По умолчанию: 5

Допустимые значения: 0 <= X

- **settings.passwordPolicy.rotation**

объект

Настройки ротации паролей.

- **settings.passwordPolicy.rotation.interval**

строка

Обязательный параметр

Интервал времени, после которого пароль необходимо изменить. Задается в секундах, минутах, часах или днях: 10s, 5m, 1h, 3d.

Шаблон: `^(?:([1-9]|[1-9][0-9]+)d)?(?:([1-9]|[1-9][0-9]+)h)?(?:([1-9]|[1-9][0-9]+)m)?(?:([1-9]|[1-9][0-9]+)s)?$`

- **settings.publishAPI**

объект

Настройки публикации API-сервера Kubernetes через Ingress для обеспечения его публичного доступа.

- **settings.publishAPI.addKubeconfigGeneratorEntry**

булевый

Если указать `false`, будет удалена запись в kubeconfig-generator.

По умолчанию: `true`

- **settings.publishAPI.enabled**

булевый

Если указать `true`, в namespace `d8-user-authn` кластера будет создан Ingress-ресурс, который откроет публичный доступ к API-серверу.

По умолчанию: `false`

- **settings.publishAPI.https**

объект

Режим работы HTTPS для Ingress API-сервера.

Примеры:

```
https:
  mode: SelfSigned
```

```
https:
  mode: Global
  global:
    kubeconfigGeneratorMasterCA: plainstring
```

- **settings.publishAPI.https.global**

объект

Дополнительный параметр для режима `Global`.

- **settings.publishAPI.https.global.kubeconfigGeneratorMasterCA**

строка

Если перед Ingress-контроллером есть внешний балансировщик, который терминирует HTTPS-трафик с использованием непубличного сертификата, укажите цепочку CA в этом параметре. Она будет

добавлена в сгенерированные файлы конфигурации kubect1.

Если вы используете в кластере сертификаты, выдаваемые с помощью модуля `cert-manager` и Let's Encrypt, следует в качестве значения установить пустую строку `" "`.

В качестве CA допускается указать непосредственно сертификат внешнего балансировщика. В таком случае нужно помнить, что обновление сертификата на балансировщике повредит ранее сгенерированные файлы конфигурации kubect1.

■ **settings.publishAPI.https.mode**

строка

Режим выдачи сертификатов для данного Ingress-ресурса.

В случае использования режима `SelfSigned`, для Ingress-ресурса будет выпущен сертификат подписанный CA.

Получить выпущенный сертификат можно следующей командой: `d8 k -n d8-user-authn get secrets kubernetes-api-ca-key-pair -oyaml`.

В случае использования режима `Global` будут применены политики из глобальной настройки `global.modules.https.mode`. То есть если в глобальной настройке стоит режим `CertManager` с `ClusterIssuer letsencrypt`, для Ingress-ресурса будет заказан сертификат Let's Encrypt.

По умолчанию: `SelfSigned`

Допустимые значения: `SelfSigned`, `Global`

■ **settings.publishAPI.ingressClass**

строка

Ingress-класс, который будет использован для публикации API Kubernetes через Ingress.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ **settings.publishAPI.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено подключение к API-серверу.

■ **Элемент массива**

строка

Шаблон: `^(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|0-9)))?$`

○ **settings.refreshTokenAbsoluteLifetime**

строка

Максимальное время жизни refresh-токенов (соответствует настройке `Dex expiry.refreshTokens.absoluteLifetime`).

Этот параметр управляет продолжительностью пользовательских сессий. По истечении этого периода пользователи должны будут пройти повторную аутентификацию, даже при наличии активных refresh-токенов.

Задается в виде строки с указанием часов, минут и секунд: `30m`, `20s`, `2h30m10s`, `24h`.

По умолчанию: `4380h`

Шаблон: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

○ **settings.staticUsers2FA**

объект

■ **settings.staticUsers2FA.enabled**

булевый

Обязательный параметр

Если указано `true`, статические пользователи должны будут использовать двухфакторную аутентификацию (2FA) при входе в систему. Эта опция полезна для повышения безопасности, так как требует дополнительного шага проверки при авторизации.

По умолчанию: `false`

- **settings.staticUsers2FA.issuerName**

строка

Имя издателя для токенов двухфакторной аутентификации (2FA). Это имя отображается пользователям в приложении для 2FA (например, Google Authenticator). Используется для идентификации источника токенов и может быть настроено в соответствии с вашими предпочтениями.

По умолчанию: `Deckhouse`

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

Пример:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**

строка

- **settings.tolerations.operator**

строка

- **settings.tolerations.tolerationSeconds**

целочисленный

- **settings.tolerations.value**

строка

Автоматический деплой `oauth2-proxy` в namespace вашего приложения и подключение его к Dex происходят при создании custom resource `DexAuthenticator`.

Важно! Так как использование OpenID Connect по протоколу HTTP является слишком значительной угрозой безопасности (что подтверждается, например, тем, что Kubernetes API-сервер не поддерживает работу с OIDC по HTTP), данный модуль можно установить только при включенном HTTPS (`https.mode` выставить в отличное от `Disabled` значение или на уровне кластера, или в самом модуле).

Важно! При включении данного модуля аутентификация во всех веб-интерфейсах перестанет использовать HTTP Basic Auth и переключится на Dex (который, в свою очередь, будет использовать настроенные вами внешние провайдеры). Для настройки `kubectl` необходимо перейти по адресу `https://kubecfg.<modules.publicDomainTemplate>/`, авторизоваться в настроенном внешнем провайдере и скопировать shell-команды к себе в консоль.

Важно! Для работы аутентификации в dashboard и `kubectl` требуется донастройка API-сервера. Для автоматизации этого процесса реализован модуль `control-plane-manager`, который включен по умолчанию.

Custom Resources

i Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

DexAuthenticator

Short names: dex-auth

Scope: Namespaced

- v1
- v2alpha1
- v1alpha1

При появлении объекта `DexAuthenticator` в пространстве имен будут созданы:

- Deployment с OAuth2-Proxy и Redis;
- Сервис (Service), ведущий на OAuth2-Proxy;
- Ingress-ресурс, который принимает запросы по адресу `https://<applicationDomain>/dex-authenticator` и отправляет их в сторону сервиса;
- HTTPRoute (опционально), который принимает запросы по адресу `https://<applicationDomain>/dex-authenticator` и отправляет их в сторону сервиса;
- Секреты, необходимые для доступа к Dex.

Внимание. DexAuthenticator работает только по HTTPS. Ingress приложения должен иметь настроенный TLS.

Внимание. При перезапуске пода с OAuth2-Proxy, с помощью refresh-токена в память Redis будут получены и сохранены актуальные Access Token и ID Token.

Пример:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
    - everyone
    - admins
  whitelistSourceRanges:
    - 1.1.1.1/32
    - 192.168.0.0/24
  additionalApplications:
    - domain: additional-app-name.kube.my-domain.com
      ingressSecretName: ingress-tls
      ingressClassName: nginx
      signOutURL: "/logout"
      whitelistSourceRanges:
        - 2.2.2.2/32
```

- **spec**
объект

Обязательный параметр

- **spec.additionalApplications**
массив объектов

Список дополнительных приложений, для которых необходима аутентификация.

■ **spec.additionalApplications.domain**

строка

Обязательный параметр

Домен приложения (для использования в Ingress-ресурсе), запросы на который будут перенаправлены в Dex для аутентификации.



Внимание. Укажите DNS-имя, соответствующее требованиям RFC 1123 (subdomain), без HTTP-схемы. Использование IP-адресов не допускается.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ **spec.additionalApplications.ingressClassName**

строка

Обязательный параметр

Название Ingress-класса, которое будет использоваться в Ingress-ресурсе (должно совпадать с названием Ingress-класса для домена приложения).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

■ **spec.additionalApplications.ingressSecretName**

строка

Имя секрета (Secret) с TLS-сертификатом для домена приложения, который используется в Ingress-ресурсе приложения. Секрет должен обязательно находиться в том же пространстве имен, что и DexAuthenticator.

Шаблон: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$)`

■ **spec.additionalApplications.signOutURL**

строка

URL для завершения сеанса аутентификации.

Используется в приложении для направления запросов на “Выход” (завершение сеанса). Для указанного URL будет создан отдельный Ingress-ресурс, запросы на который будут перенаправляться в `dex-authenticator`.

Значение не должно содержать переводы строк.

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

■ **spec.additionalApplications.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

■ **Элемент массива**

строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

○ **spec.allowedEmails**

массив строк

Email'ы пользователей, которым разрешено проходить аутентификацию.

По умолчанию разрешено всем пользователям.

Значения не должны содержать переводы строк.

По умолчанию: All emails are allowed.

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 320

- **spec.allowedGroups**

массив строк

Группы, пользователям которых разрешено проходить аутентификацию.

По умолчанию разрешено всем группам.

Значения не должны содержать переводы строк.

По умолчанию: `All groups are allowed.`

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 512

- **spec.applicationDomain**

строка

Обязательный параметр

Адрес приложения (домен DNS для Ingress-ресурса), для которого необходима аутентификация.



Внимание. Укажите DNS-имя, соответствующее требованиям RFC 1123 (subdomain), без HTTP-схемы. Использование IP-адресов не допускается.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
applicationDomain: my-app.domain.com
```

- **spec.applicationIngressCertificateSecretName**

строка

Имя секрета (Secret) с TLS-сертификатом для домена приложения, который используется в Ingress-ресурсе приложения. Секрет должен обязательно находиться в том же пространстве имен, что и DexAuthenticator.

Шаблон: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Пример:

```
applicationIngressCertificateSecretName: ingress-tls
```

- **spec.applicationIngressClassName**

строка

Обязательный параметр

Название Ingress-класса, которое будет использоваться в Ingress-ресурсе (должно совпадать с названием Ingress-класса для домена приложения).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
applicationIngressClassName: nginx
```

- **spec.gatewayAPI**

объект

Набор параметров для настройки создаваемого объекта HTTPRoute.



Внимание. Используется только в кластерах с поддержкой Gateway API.



Внимание. Должен быть указан тот же объект ListenerSet, который используется для домена приложения.

■ **spec.gatewayAPI.applicationHTTPRouteListenerSetName**

строка

Обязательный параметр

Имя объекта ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 253

Пример:

```
applicationHTTPRouteListenerSetName: default-listenerset
```

■ **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**

строка

Имя неймспейса, в котором располагается объект ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 63

Пример:

```
applicationHTTPRouteListenerSetNamespace: default
```

■ **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**

строка

Имя секции объекта ListenerSet из Gateway API, на которую будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 253

Пример:

```
applicationHTTPRouteListenerSetSectionName: default-listener
```

○ **spec.highAvailability**

булевый

Включает режим высокой доступности для экземпляра `dex-authenticator`.

При включении этого режима будет развернуто несколько реплик подов для обеспечения отказоустойчивости.

По умолчанию: `false`

○ **spec.keepUsersLoggedInFor**

строка

Максимальное время бездействия пользователя, в течение которого его сессия остаётся активной.

Задается в виде строки с указанием часов, минут и секунд: `30m`, `20s`, `2h30m10s`, `24h`.

Указанное значение не может быть меньше или равно значению параметра `idTokenTTL`, заданному в настройках модуля. Если указать значение, меньшее или равное `idTokenTTL`, фактическое время до выхода пользователя из системы составит `idTokenTTL + 1s`.

Чтобы пользователь выходил из системы раньше значения `idTokenTTL` по умолчанию, уменьшите также значение параметра `idTokenTTL` в настройках модуля.

По умолчанию: `168h`

Пример:

```
keepUsersLoggedInFor: 24h
```

- **spec.nodeSelector**

объект

Определяет `nodeSelector` для подов `dex-authenticator`.

Формат: стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- **spec.resources**

объект

Ресурсы контейнеров (CPU/память) — `requests/limits`.

Если поле задано, применяются указанные значения, а VPA для этого экземпляра отключается. Если поле не задано, поведение не меняется — при включённом модуле VPA ресурсы управляются VPA, при выключенном — используются значения по умолчанию.

Ресурсы можно задать напрямую для контейнера `dex-authenticator` (для обратной совместимости) либо во вложенных объектах — отдельно для каждого контейнера.

- **spec.resources.limits**

объект

Верхняя граница ресурсов контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis**

объект

Ресурсы контейнера `redis` (CPU/память) — `requests/limits`.

- **spec.resources.redis.limits**

объект

Верхняя граница ресурсов контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis.requests**

объект

Гарантированно выделяемые ресурсы контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.resources.requests**

объект

Гарантированно выделяемые ресурсы контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.sendAuthorizationHeader**

булевый

Флаг, который отвечает за отправку конечному приложению заголовка `Authorization: Bearer`.

Внимание. Включайте, только если приложение умеет обрабатывать такой заголовок.

- **spec.signOutURL**

строка

URL для завершения сеанса аутентификации.

Используется в приложении для направления запросов на «Выход» (завершение сеанса). Для указанного URL будет создан отдельный Ingress-ресурс, запросы на который будут перенаправляться в `dex-authenticator`.

Значение не должно содержать переводы строк.

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

- **spec.tolerations**

массив объектов

Определяет `tolerations` для подов `dex-authenticator`.

Формат: стандартный список `toleration`. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**

строка

Определяет, какому `effect`'у `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому `effect`.

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**

строка

Определяет, какому ключу (`key`) `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому ключу.

Если ключ (`key`) не указан (пуст), `operator` должен быть `Exists`, что подразумевает соответствие любым `value` и `key`.

- **spec.tolerations.operator**

строка

Определяет отношение ключа (`key`) к значению (`value`) — должен ли ключ быть равен (`Equal`) значению или допустимо существование ключа с любым значением.

Указание `Exists` означает, что значение `value` может быть любым, чтобы под с указанным `toleration` был размещён на узле с соответствующим `taint`.

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**

целочисленный

Определяет период времени в секундах, в течение которого планировщик должен ждать, прежде чем выселить (`evict`) под с узла, если `toleration` больше не соответствует `taint` (справедливо **только** для `effect` `NoExecute`, иначе игнорируется).

Если **не установлено**, тогда под не будет выселен с узла, если `toleration` больше не соответствует `taint`. Если **установлен в ноль** (или отрицателен), под будет выселен с узла немедленно, если `toleration` больше не соответствует `taint`.

По умолчанию используется значение «не установлено».

- **spec.tolerations.value**

строка

Значение, которому должен соответствовать `toleration`.

Должно быть пустым, если `operator` имеет параметр `Exists`.

- **spec.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec**
объект

Обязательный параметр

- **spec.allowedEmails**
массив строк

A list of emails of users are allowed to connect to the client.

By default, all users can connect.

Values must not contain line breaks.

По умолчанию: `All users are allowed.`

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: `320`

- **spec.allowedGroups**
массив строк

Группы, пользователям которых разрешено проходить аутентификацию.

По умолчанию разрешено всем группам.

Значения не должны содержать переводы строк.

По умолчанию: `All groups are allowed.`

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: `512`

- **spec.applications**
массив объектов

Обязательный параметр

Список приложений, для которых необходима аутентификация.

- **spec.applications.domain**
строка

Обязательный параметр

Домен приложения (для использования в Ingress-ресурсе), запросы на который будут перенаправлены в Dex для аутентификации.

i

Внимание. Укажите DNS-имя, соответствующее требованиям RFC 1123 (subdomain), без HTTP-схемы. Использование IP-адресов не допускается.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.applications.gatewayAPI**
объект

Набор параметров для настройки создаваемого объекта HTTPRoute.



Внимание. Используется только в кластерах с поддержкой Gateway API.



Внимание. Должен быть указан тот же объект ListenerSet, который используется для домена приложения.

■ **spec.applications.gatewayAPI.httpRouteListenerSetName**

строка

Обязательный параметр

Имя объекта ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Максимальная длина: 253

Пример:

```
httpRouteListenerSetName: default-listener-set
```

■ **spec.applications.gatewayAPI.httpRouteListenerSetNamespace**

строка

Имя неймспейса, в котором располагается объект ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Максимальная длина: 63

Пример:

```
httpRouteListenerSetNamespace: default
```

■ **spec.applications.gatewayAPI.httpRouteListenerSetSectionName**

строка

Имя секции объекта ListenerSet из Gateway API, на которую будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Максимальная длина: 253

Пример:

```
httpRouteListenerSetSectionName: web-listener
```

■ **spec.applications.ingressClassName**

строка

Обязательный параметр

Название Ingress-класса, которое будет использоваться в Ingress-ресурсе (должно совпадать с названием Ingress-класса для домена приложения).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

■ **spec.applications.ingressSecretName**

строка

Имя секрета (Secret) с TLS-сертификатом для домена приложения, который используется в Ingress-ресурсе приложения. Секрет должен обязательно находиться в том же пространстве имен, что и DexAuthenticator.

Шаблон: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.applications.signOutURL**

строка

URL для завершения сеанса аутентификации.

Используется в приложении для направления запросов на «Выход» (завершение сеанса). Для указанного URL будет создан отдельный Ingress-ресурс, запросы на который будут перенаправляться в `dex-authenticator`.

Значение не должно содержать переводы строк.

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

- **spec.applications.whitelistSourceRanges**

массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

- **Элемент массива**

строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.highAvailability**

булевый

Enables high availability mode for `dex-authenticator`.

When this mode is enabled, multiple pod replicas will be deployed to ensure fault tolerance.

По умолчанию: `false`

- **spec.keepUsersLoggedInFor**

строка

Максимальное время бездействия пользователя, в течение которого его сессия остаётся активной.

Задается в виде строки с указанием часов, минут и секунд: `30m`, `20s`, `2h30m10s`, `24h`.

Указанное значение не может быть меньше или равно значению параметра `idTokenTTL`, заданному в настройках модуля. Если указать значение, меньшее или равное `idTokenTTL`, фактическое время до выхода пользователя из системы составит `idTokenTTL + 1s`.

Чтобы пользователь выходил из системы раньше значения `idTokenTTL` по умолчанию, уменьшите также значение параметра `idTokenTTL` в настройках модуля.

По умолчанию: `168h`

- **spec.nodeSelector**

объект

Определяет `nodeSelector` для подов `dex-authenticator`.

Формат: стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- **spec.resources**

объект

Ресурсы контейнеров (CPU/память) — `requests/limits`.

Если поле задано, применяются указанные значения, а VPA для этого экземпляра отключается. Если поле не задано, поведение не меняется — при включённом модуле VPA ресурсы управляются VPA, при выключенном — используются значения по умолчанию.

Ресурсы можно задать напрямую для контейнера `dex-authenticator` (для обратной совместимости) либо во вложенных объектах — отдельно для каждого контейнера.

- **spec.resources.limits**

объект

Верхняя граница ресурсов контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis**

объект

Ресурсы контейнера `redis` (CPU/память) — requests/limits.

- **spec.resources.redis.limits**

объект

Верхняя граница ресурсов контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis.requests**

объект

Гарантированно выделяемые ресурсы контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.resources.requests**

объект

Гарантированно выделяемые ресурсы контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.sendAuthorizationHeader**

булевый

Флаг, который отвечает за отправку конечному приложению заголовка `Authorization: Bearer`.

Внимание. Включайте, только если приложение умеет обрабатывать такой заголовок.

- **spec.tolerations**

массив объектов

Определяет tolerations для подов `dex-authenticator`.

Формат: стандартный список toleration. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**

строка

Определяет, какому effect'у taint'a соответствует описываемый toleration. Пустой подразумевает соответствие любому effect.

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**

строка

Определяет, какому ключу (key) taint'a соответствует описываемый toleration. Пустой подразумевает соответствие любому ключу.

Если ключ (key) не указан (пуст), `operator` должен быть `Exists`, что подразумевает соответствие любым `value` и `key`.

- **spec.tolerations.operator**

строка

Определяет отношение ключа (key) к значению (value) — должен ли ключ быть равен (`Equal`) значению или допустимо существование ключа с любым значением.

Указание `Exists` означает, что значение `value` может быть любым, чтобы под с указанным toleration был размещён на узле с соответствующим taint.

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

■ **spec.tolerations.tolerationSeconds**

целочисленный

Определяет период времени в секундах, в течение которого планировщик должен ждать, прежде чем выселить (evict) под с узла, если toleration больше не соответствует taint (справедливо **только** для effect `NoExecute`, иначе игнорируется).

Если **не установлено**, тогда под не будет выселен с узла, если toleration больше не соответствует taint. Если **установлен в ноль** (или отрицателен), под будет выселен с узла немедленно, если toleration больше не соответствует taint.

По умолчанию используется значение «не установлено».

■ **spec.tolerations.value**

строка

Значение, которому должен удовлетворять toleration.

Должно быть пустым, если `operator` имеет параметр `Exists`.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

При появлении объекта `DexAuthenticator` в пространстве имен будут созданы:

- Deployment с OAuth2-Proxy и Redis;
- Сервис (Service), ведущий на OAuth2-Proxy;
- Ingress-ресурс, который принимает запросы по адресу `https://<applicationDomain>/dex-authenticator` и отправляет их в сторону сервиса;
- HTTPRoute (опционально), который принимает запросы по адресу `https://<applicationDomain>/dex-authenticator` и отправляет их в сторону сервиса;
- Секреты, необходимые для доступа к Dex.

Внимание. DexAuthenticator работает только по HTTPS. Ingress приложения должен иметь настроенный TLS.

Внимание. При перезапуске пода с OAuth2-Proxy, с помощью refresh-токена в память Redis будут получены и сохранены актуальные Access Token и ID Token.

Пример:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
    - everyone
    - admins
  whitelistSourceRanges:
    - 1.1.1.1/32
    - 192.168.0.0/24
  additionalApplications:
    - domain: additional-app-name.kube.my-domain.com
      ingressSecretName: ingress-tls
      ingressClassName: nginx
      signOutURL: "/logout"
      whitelistSourceRanges:
        - 2.2.2.2/32
```


- **spec**
объект

Обязательный параметр

- **spec.additionalApplications**
массив объектов

Список дополнительных приложений, для которых необходима аутентификация.

- **spec.additionalApplications.domain**
строка

Обязательный параметр

Домен приложения (для использования в Ingress-ресурсе), запросы на который будут перенаправлены в Dex для аутентификации.

i

Внимание. Укажите DNS-имя, соответствующее требованиям RFC 1123 (subdomain), без HTTP-схемы. Использование IP-адресов не допускается.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressClassName**
строка

Обязательный параметр

Название Ingress-класса, которое будет использоваться в Ingress-ресурсе (должно совпадать с названием Ingress-класса для домена приложения).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressSecretName**
строка

Имя секрета (Secret) с TLS-сертификатом для домена приложения, который используется в Ingress-ресурсе приложения. Секрет должен обязательно находиться в том же пространстве имен, что и DexAuthenticator.

Шаблон: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$)`

- **spec.additionalApplications.signOutURL**
строка

URL для завершения сеанса аутентификации.

Используется в приложении для направления запросов на “Выход” (завершение сеанса). Для указанного URL будет создан отдельный Ingress-ресурс, запросы на который будут перенаправляться в dex-authenticator.

Значение не должно содержать переводы строк.

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

- **spec.additionalApplications.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.allowedEmails**

массив строк

Email'ы пользователей, которым разрешено проходить аутентификацию.

По умолчанию разрешено всем пользователям.

Значения не должны содержать переводы строк.

По умолчанию: `All emails are allowed.`

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: `320`

- **spec.allowedGroups**

массив строк

Группы, пользователям которых разрешено проходить аутентификацию.

По умолчанию разрешено всем группам.

Значения не должны содержать переводы строк.

По умолчанию: `All groups are allowed.`

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: `512`

- **spec.applicationDomain**

строка

Обязательный параметр

Адрес приложения (домен DNS для Ingress-ресурса), для которого необходима аутентификация.

i

Внимание. Укажите DNS-имя, соответствующее требованиям RFC 1123 (subdomain), без HTTP-схемы. Использование IP-адресов не допускается.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
applicationDomain: my-app.domain.com
```

- **spec.applicationIngressCertificateSecretName**

строка

Имя секрета (Secret) с TLS-сертификатом для домена приложения, который используется в Ingress-ресурсе приложения. Секрет должен обязательно находиться в том же пространстве имен, что и DexAuthenticator.

Шаблон: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

Пример:

```
applicationIngressCertificateSecretName: ingress-tls
```

- **spec.applicationIngressClassName**

строка

Обязательный параметр

Название Ingress-класса, которое будет использоваться в Ingress-ресурсе (должно совпадать с названием Ingress-класса для домена приложения).

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Пример:

```
applicationIngressClassName: nginx
```

- **spec.gatewayAPI**
объект

Набор параметров для настройки создаваемого объекта HTTPRoute.

i

Внимание. Используется только в кластерах с поддержкой Gateway API.

i

Внимание. Должен быть указан тот же объект ListenerSet, который используется для домена приложения.

- **spec.gatewayAPI.applicationHTTPRouteListenerSetName**
строка

Обязательный параметр

Имя объекта ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 253

Пример:

```
applicationHTTPRouteListenerSetName: default-listenerset
```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**
строка

Имя неймспейса, в котором располагается объект ListenerSet из Gateway API, на который будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 63

Пример:

```
applicationHTTPRouteListenerSetNamespace: default
```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**
строка

Имя секции объекта ListenerSet из Gateway API, на которую будет ссылаться создаваемый HTTPRoute.

Шаблон: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

Максимальная длина: 253

Пример:

```
applicationHTTPRouteListenerSetSectionName: default-listener
```

- **spec.highAvailability**
булевый

Включает режим высокой доступности для экземпляра `dex-authenticator`.

При включении этого режима будет развернуто несколько реплик подов для обеспечения отказоустойчивости.

По умолчанию: `false`

- **spec.keepUsersLoggedInFor**
строка

Максимальное время бездействия пользователя, в течение которого его сессия остаётся активной.

Задается в виде строки с указанием часов, минут и секунд: `30m`, `20s`, `2h30m10s`, `24h`.

Указанное значение не может быть меньше или равно значению параметра `idTokenTTL`, заданному в настройках модуля. Если указать значение, меньшее или равное `idTokenTTL`, фактическое время до выхода пользователя из системы составит `idTokenTTL + 1s`.

Чтобы пользователь выходил из системы раньше значения `idTokenTTL` по умолчанию, уменьшите также значение параметра `idTokenTTL` в настройках модуля.

По умолчанию: `168h`

Пример:

```
keepUsersLoggedInFor: 24h
```

- **spec.nodeSelector**
объект

Определяет `nodeSelector` для подов `dex-authenticator`.

Формат: стандартный список `nodeSelector`. Поды инстанса унаследуют это поле как есть.

- **spec.resources**
объект

Ресурсы контейнеров (CPU/память) — `requests/limits`.

Если поле задано, применяются указанные значения, а VPA для этого экземпляра отключается. Если поле не задано, поведение не меняется — при включённом модуле VPA ресурсы управляются VPA, при выключенном — используются значения по умолчанию.

Ресурсы можно задать напрямую для контейнера `dex-authenticator` (для обратной совместимости) либо во вложенных объектах — отдельно для каждого контейнера.

- **spec.resources.limits**
объект

Верхняя граница ресурсов контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis**
объект

Ресурсы контейнера `redis` (CPU/память) — `requests/limits`.

- **spec.resources.redis.limits**
объект

Верхняя граница ресурсов контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `200m`, `256Mi`.

- **spec.resources.redis.requests**
объект

Гарантированно выделяемые ресурсы контейнера `redis` (CPU/память).

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.resources.requests**
объект

Гарантированно выделяемые ресурсы контейнера `dex-authenticator` (CPU/память). Поле поддерживается для обратной совместимости.

Значения указываются в формате Kubernetes, например: `100m`, `128Mi`.

- **spec.sendAuthorizationHeader**

булевый

Флаг, который отвечает за отправку конечному приложению заголовка `Authorization: Bearer`.

Внимание. Включайте, только если приложение умеет обрабатывать такой заголовок.

- **spec.signOutURL**

строка

URL для завершения сеанса аутентификации.

Используется в приложении для направления запросов на «Выход» (завершение сеанса). Для указанного URL будет создан отдельный Ingress-ресурс, запросы на который будут перенаправляться в `dex-authenticator`.

Значение не должно содержать переводы строк.

Шаблон: `^[^\r\n]*$`

Максимальная длина: `2048`

- **spec.tolerations**

массив объектов

Определяет `tolerations` для подов `dex-authenticator`.

Формат: стандартный список `toleration`. Поды инстанса унаследуют это поле как есть.

- **spec.tolerations.effect**

строка

Определяет, какому `effect`'у `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому `effect`.

Допустимые значения: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**

строка

Определяет, какому ключу (`key`) `taint`'а соответствует описываемый `toleration`. Пустой подразумевает соответствие любому ключу.

Если ключ (`key`) не указан (пуст), `operator` должен быть `Exists`, что подразумевает соответствие любым `value` и `key`.

- **spec.tolerations.operator**

строка

Определяет отношение ключа (`key`) к значению (`value`) — должен ли ключ быть равен (`Equal`) значению или допустимо существование ключа с любым значением.

Указание `Exists` означает, что значение `value` может быть любым, чтобы под с указанным `toleration` был размещён на узле с соответствующим `taint`.

По умолчанию: `Equal`

Допустимые значения: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**

целочисленный

Определяет период времени в секундах, в течение которого планировщик должен ждать, прежде чем выселить (`evict`) под с узла, если `toleration` больше не соответствует `taint` (справедливо **только** для `effect` `NoExecute`, иначе игнорируется).

Если **не установлено**, тогда под не будет выселен с узла, если `toleration` больше не соответствует `taint`. Если **установлен в ноль** (или отрицателен), под будет выселен с узла немедленно, если `toleration` больше не соответствует `taint`.

По умолчанию используется значение «не установлено».

- **spec.tolerations.value**

строка

Значение, которому должен соответствовать `toleration`.

Должно быть пустым, если `operator` имеет параметр `Exists`.

- **spec.whitelistSourceRanges**
массив строк

Список адресов в формате CIDR, которым разрешено проходить аутентификацию. Если параметр не указан, аутентификацию разрешено проходить без ограничения по IP-адресу.

Пример:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

- **Элемент массива**
строка

Шаблон: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

DexClient

Scope: Namespaced

- v1
- v1alpha1

Позволяет приложениям, поддерживающим DC-аутентификацию, взаимодействовать с Dex.

После появления в кластере объекта `DexClient`:

- в Dex будет зарегистрирован клиент с идентификатором (**clientID**) `dex-client-<NAME>@<NAMESPACE>`, где `<NAME>` и `<NAMESPACE>` — `metadata.name` и `metadata.namespace` объекта `DexClient` соответственно;
- в соответствующем namespace будет создан Secret `dex-client-<NAME>` (где `<NAME>` — `metadata.name` объекта `DexClient`), содержащий пароль доступа к клиенту (**clientSecret**).

Пример использования...

- **spec**
объект

Обязательный параметр

- **spec.allowedEmails**
массив строк

Email'ы пользователей, которым разрешено проходить аутентификацию.

По умолчанию разрешено всем пользователям.

Значения не должны содержать переводы строк.

По умолчанию: `All users are allowed.`

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 320

- **spec.allowedGroups**
массив строк

Группы, пользователям которых разрешено проходить аутентификацию.

По умолчанию разрешено всем группам.

Значения не должны содержать переводы строк.

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 512

- **spec.redirectURIs**
массив строк

Список адресов, на которые допустимо редиректить Dex'у после успешного прохождения аутентификации.

Значения не должны содержать переводы строк.

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

- **spec.secretMetadata**
объект

Список лейблов и аннотаций, которые будут перенесены в метаданные секретов ресурса DexClient.

- **spec.secretMetadata.annotations**
объект
- **spec.secretMetadata.labels**
объект

- **spec.trustedPeers**
массив строк

ID клиентов, которым позволена cross-аутентификация.

Значения не должны содержать переводы строк.

Подробнее...

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 256

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Позволяет приложениям, поддерживающим DC-аутентификацию, взаимодействовать с Dex.

После появления в кластере объекта DexClient:

- в Dex будет зарегистрирован клиент с идентификатором (**clientID**) `dex-client-<NAME>@<NAMESPACE>`, где `<NAME>` и `<NAMESPACE>` — `metadata.name` и `metadata.namespace` объекта DexClient соответственно;
- в соответствующем namespace будет создан Secret `dex-client-<NAME>` (где `<NAME>` — `metadata.name` объекта DexClient), содержащий пароль доступа к клиенту (**clientSecret**).

Пример использования...

- **spec**
объект

Обязательный параметр

- **spec.allowedEmails**
массив строк

Email'ы пользователей, которым разрешено проходить аутентификацию.

По умолчанию разрешено всем пользователям.

Значения не должны содержать переводы строк.

По умолчанию: `All users are allowed.`

- **Элемент массива**
строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 320

- **spec.allowedGroups**

массив строк

Группы, пользователям которых разрешено проходить аутентификацию.

По умолчанию разрешено всем группам.

Значения не должны содержать переводы строк.

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 512

- **spec.redirectURIs**

массив строк

Список адресов, на которые допустимо редиректить Dex'у после успешного прохождения аутентификации.

Значения не должны содержать переводы строк.

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 2048

- **spec.secretMetadata**

объект

Список лейблов и аннотаций, которые будут перенесены в метаданные секретов ресурса DexClient.

- **spec.secretMetadata.annotations**

объект

- **spec.secretMetadata.labels**

объект

- **spec.trustedPeers**

массив строк

ID клиентов, которым позволена cross-аутентификация.

Значения не должны содержать переводы строк.

Подробнее...

- **Элемент массива**

строка

Шаблон: `^[^\r\n]*$`

Максимальная длина: 256

DexProvider

Scope: Cluster

- v1
- v1alpha1

Описывает конфигурацию подключения стороннего провайдера.

С его помощью можно гибко настроить интеграцию каталога учетных записей с Kubernetes.

Пример использования...

Внимание. Запрещено использовать пользователей и группы с префиксом `system:`. Аутентификация таких пользователей или участников этих групп будет отклонена, а в логах `kube-apiserver` появится соответствующее предупреждение.

- **spec**
объект

Обязательный параметр

- **spec.bitbucketCloud**
объект

Параметры провайдера Bitbucket Cloud (можно указывать, только если `type: BitbucketCloud`).

- **spec.bitbucketCloud.clientID**
строка

Обязательный параметр

ID приложения, созданного в Bitbucket Cloud (Key).

- **spec.bitbucketCloud.clientSecret**
строка

Обязательный параметр

Secret приложения, созданного в Bitbucket Cloud (Secret).

- **spec.bitbucketCloud.includeTeamGroups**
булевый

Включает в список команд все группы команды, в которых состоит пользователь.

Пример групп пользователя с включенной опцией:

```
groups=["my_team", "my_team/administrators", "my_team/members"]
```

По умолчанию: `false`

- **spec.bitbucketCloud.teams**
массив строк

Список-фильтр команд, допустимых для приема из Bitbucket Cloud.

Токен пользователя будет содержать пересечение множеств команд из Bitbucket Cloud и команд из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Токен будет содержать команды пользователя в claim `groups`, как и у других провайдеров.

- **spec.crowd**
объект

Параметры провайдера Crowd (можно указывать, только если `type: Crowd`).

- **spec.crowd.baseURL**
строка

Обязательный параметр

Адрес Crowd.

Пример:

```
baseURL: https://crowd.example.com/crowd
```

- **spec.crowd.clientID**
строка

Обязательный параметр

ID приложения, созданного в Crowd (Application Name).

- **spec.crowd.clientSecret**
строка

Обязательный параметр

Пароль приложения, созданного в Crowd (Password).

- **spec.crowd.enableBasicAuth**
булевый

Включает возможность базовой аутентификации (Basic Authentication) для

Kubernetes API server.

В качестве учетных данных используются логин и пароль пользователя Crowd (из приложения, настроенного для этого провайдера).

Ограничения:

- работает только при включенном `publishAPI`;
- может быть включено только для одного провайдера аутентификации типа OIDC или Crowd в кластере.

Данные аутентификации и группы, полученные от провайдера аутентификации, кешируются на 10 секунд.

- **spec.crowd.groups**

массив строк

Список-фильтр групп, допустимых для приема из Crowd.

Токен пользователя будет содержать пересечение множеств групп из Crowd и групп из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все группы из Crowd.

- **spec.crowd.usernamePrompt**

строка

Строка, которая будет отображаться возле поля для имени пользователя в форме ввода логина и пароля.

По умолчанию: `Crowd username`

- **spec.displayName**

строка

Обязательный параметр

Имя провайдера, которое будет отображено на странице выбора провайдера для аутентификации.

Если настроен всего один провайдер, страница выбора провайдера показываться не будет.

- **spec.enabled**

булевый

When false, this provider is ignored by Dex config renderer.

По умолчанию: `true`

- **spec.github**

объект

Параметры провайдера GitHub (можно указывать, только если `type: Github`).

- **spec.github.clientID**

строка

Обязательный параметр

ID организации на GitHub.

- **spec.github.clientSecret**

строка

Обязательный параметр

Secret организации на GitHub.

- **spec.github.orgs**

массив объектов

Массив названий организаций в GitHub.

- **spec.github.orgs.name**

строка

Обязательный параметр

Название организации.

- **spec.github.orgs.teams**

массив строк

Список-фильтр команд, допустимых для приема из GitHub.

Токен пользователя будет содержать пересечение множеств команд из GitHub и команд из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все команды из GitHub.

- **spec.github.teamNameField**

строка

Формат команд, которые будут получены из GitHub.

Если в организации `асме` есть группа `Site Reliability Engineers`, в случае:

- `Name` будет получена группа с именем `['асме:Site Reliability Engineers']`;
- `Slug` будет получена группа с именем `['асме:site-reliability-engineers']`;
- `Both` будут получены группы с именами `['асме:Site Reliability Engineers', 'асме:site-reliability-engineers']`.

По умолчанию: `Name`

Допустимые значения: `Name`, `Slug`, `Both`

- **spec.github.useLoginAsID**

булевый

Позволяет вместо использования внутреннего GitHub ID использовать имя пользователя.

- **spec.gitlab**

объект

Параметры провайдера GitLab (можно указывать, только если `type: Gitlab`).

- **spec.gitlab.baseURL**

строка

Адрес GitLab.

Пример:

```
baseURL: https://gitlab.example.com
```

- **spec.gitlab.clientID**

строка

Обязательный параметр

ID приложения, созданного в GitLab (Application ID).

- **spec.gitlab.clientSecret**

строка

Обязательный параметр

Secret приложения, созданного в GitLab (Secret).

- **spec.gitlab.groups**

массив строк

Список-фильтр групп (**пути групп — path**, а не имена), допустимых для приема из GitLab.

Токен пользователя будет содержать пересечение множеств групп из GitLab и групп из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все группы из GitLab.

- **spec.gitlab.rootCAData**

строка

A CA chain to validate the provider in PEM format.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.gitlab.useLoginAsID**

булевый

Позволяет вместо использования внутреннего GitLab ID, использовать имя пользователя.

- **spec.ldap**

объект

Параметры провайдера LDAP.

- **spec.ldap.bindDN**

строка

Путь до сервис-аккаунта приложения в LDAP.

Пример:

```
bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
```

- **spec.ldap.bindPW**

строка

Пароль для сервис-аккаунта приложения в LDAP.

Пример:

```
bindPW: password
```

- **spec.ldap.enableBasicAuth**

булевый

Включает возможность базовой аутентификации (Basic Authentication) для Kubernetes API server с использованием этого LDAP-провайдера.

В качестве учетных данных используются логин и пароль пользователя из LDAP. Эта функция недоступна если в кластере присутствует более одного провайдера аутентификации (IdP) типа `OIDC`, `Crowd` или `LDAP`.

Работает **только** при включенном параметре `publishAPI`.

Полученные от IdP данные аутентификации и групп сохраняются в кеш на 10 секунд.

- **spec.ldap.groupSearch**

объект

Настройки фильтра для поиска групп для указанного пользователя.

Подробнее о процессе фильтрации...

- **spec.ldap.groupSearch.baseDN**

строка

Обязательный параметр

Откуда будет начат поиск групп

Пример:

```
baseDN: cn=users,dc=example,dc=com
```

- **spec.ldap.groupSearch.filter**

строка

Фильтр для директории с группами.

Пример:

```
filter: "(objectClass=person)"
```

- **spec.ldap.groupSearch.nameAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранится уникальное имя группы.

Пример:

```
nameAttr: name
```

- **spec.ldap.groupSearch.userMatchers**

массив объектов

Обязательный параметр

Список сопоставлений атрибута имени юзера с именем группы.

- **spec.ldap.groupSearch.userMatchers.groupAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранятся имена пользователей, состоящих в группе.

Пример:

```
groupAttr: member
```

- **spec.ldap.groupSearch.userMatchers.userAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранится имя пользователя.

Пример:

```
userAttr: uid
```

- **spec.ldap.host**

строка

Обязательный параметр

Адрес и порт (опционально) LDAP-сервера.

Пример:

```
host: ldap.example.com:636
```

- **spec.ldap.insecureNoSSL**

булевый

Подключаться к каталогу LDAP не по защищенному порту.

По умолчанию: `false`

- **spec.ldap.insecureSkipVerify**

булевый

Не производить проверки подлинности провайдера с помощью TLS. Небезопасно

не производить проверку подлинности провайдера с помощью TLS. небезопасно, не рекомендуется использовать в production-окружениях.

По умолчанию: `false`

■ **spec.idap.kerberos**

объект

Опциональные настройки Kerberos (SPNEGO) SSO для LDAP.

■ **spec.idap.kerberos.enabled**

булевый

Включить поток аутентификации Kerberos SPNEGO для данного LDAP-провайдера.

По умолчанию: `false`

■ **spec.idap.kerberos.expectedRealm**

строка

Необязательный требуемый realm (без учета регистра). Если указан и не совпадает с realm билета — вход отклоняется.

■ **spec.idap.kerberos.fallbackToPassword**

булевый

Если `true`, при отсутствии/некорректном заголовке `Authorization: Negotiate` будет показана форма ввода логина/пароля вместо ответа 401 с `WWW-Authenticate: Negotiate`.

По умолчанию: `false`

■ **spec.idap.kerberos.keytabSecretName**

строка

Имя секрета в неймспейсе `d8-user-authn`, содержащего ключ `krb5.keytab`, который монтируется в под Dex.

■ **spec.idap.kerberos.krb5ConfConfigMapName**

строка

Необязательное имя ConfigMap с ключом `krb5.conf` для монтирования в `/etc/krb5.conf` (обычно не требуется для офлайн-проверки билетов).

■ **spec.idap.kerberos.krb5ConfSecretName**

строка

Необязательное имя секрета с ключом `krb5.conf` для монтирования в `/etc/krb5.conf` (обычно не требуется для офлайн-проверки билетов).

■ **spec.idap.kerberos.usernameFromPrincipal**

строка

Способ сопоставления Kerberos principal с именем пользователя в LDAP: `localpart`, `sAMAccountName` или `userPrincipalName`.

По умолчанию: `Localpart`

Допустимые значения: `Localpart`, `SAMAccountName`, `UserPrincipalName`

■ **spec.idap.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
-----BEGIN CERTIFICATE-----
MIIFaDC...
-----END CERTIFICATE-----
```

■ **spec.idap.startTLS**

булевый

Использовать STARTTLS для шифрования.

По умолчанию: `false`

■ **spec.ldap.userSearch**
объект

Обязательный параметр

Настройки фильтров пользователей, которые помогают сначала отфильтровать директории, в которых будет производиться поиск пользователей, а затем найти пользователя по полям (его имени, адресу электронной почты или отображаемому имени).

Подробнее о процессе фильтрации...

■ **spec.ldap.userSearch.baseDN**
строка

Обязательный параметр

Откуда будет начат поиск пользователей.

Пример:

```
baseDN: cn=users,dc=example,dc=com
```

■ **spec.ldap.userSearch.emailAttr**
строка

Обязательный параметр

Имя атрибута, из которого будет получен email пользователя.

Пример:

```
emailAttr: mail
```

■ **spec.ldap.userSearch.filter**
строка

Позволяет добавить фильтр для директории с пользователями.

Пример:

```
filter: "(objectClass=person)"
```

■ **spec.ldap.userSearch.idAttr**
строка

Обязательный параметр

Имя атрибута, из которого будет получен ID пользователя.

Пример:

```
idAttr: uid
```

■ **spec.ldap.userSearch.nameAttr**
строка

Атрибут отображаемого имени пользователя.

Пример:

```
nameAttr: name
```

■ **spec.ldap.userSearch.username**
строка

Обязательный параметр

Имя атрибута, из которого будет получен username пользователя.

Пример:

```
username: uid
```

- **spec.idap.usernamePrompt**

строка

Строка, которая будет отображаться возле поля для имени пользователя в форме ввода логина и пароля.

По умолчанию: `LDAP username`

Пример:

```
usernamePrompt: SSO Username
```

- **spec.oidc**

объект

Параметры провайдера OIDC (можно указывать, только если `type: 0IDC`).

- **spec.oidc.allowedGroups**

массив строк

Список групп, которые должны быть у пользователя для успешной аутентификации.

Если список содержит группу, которой у пользователя нет, аутентификация будет отклонена.

Если параметр не задан, проверка принадлежности к группам не выполняется.

- **spec.oidc.basicAuthUnsupported**

булевый

Использовать POST-запросы для общения с провайдером вместо добавления токена в Basic Authorization header.

В большинстве случаев Dex сам определяет, какой запрос ему нужно сделать, но иногда включение этого параметра может помочь.

По умолчанию: `false`

- **spec.oidc.claimMapping**

объект

Некоторые провайдеры возвращают нестандартные claim'ы (например, mail). Claim mappings помогают Dex преобразовать их в стандартные claim'ы OIDC.

Dex может преобразовать нестандартный claim в стандартный, только если id_token, полученный от OIDC-провайдера, не содержит аналогичный стандартный claim.

- **spec.oidc.claimMapping.email**

строка

Claim, который будет использован для получения email пользователя.

По умолчанию: `email`

- **spec.oidc.claimMapping.groups**

строка

Claim, который будет использован для получения групп пользователя.

По умолчанию: `groups`

- **spec.oidc.claimMapping.preferred_username**

строка

Claim, который будет использован для получения предпочтительного имени пользователя.

По умолчанию: `preferred_username`

- **spec.oidc.claimMappingOverride**

булевый

Если включено, сопоставление claim'ов (`claimMapping`) будет переопределять

стандартные claim'ы OIDC.

По умолчанию сопоставление claim'ов будет использоваться только в том случае, если стандартные claim'ы OIDC отсутствуют, например, если в `id_token` нет claim'a `email`, будет использоваться `claimMapping.email`.

По умолчанию: `false`

- **spec.oidc.clientID**

строка

Обязательный параметр

ID приложения, созданного в OIDC-провайдере.

- **spec.oidc.clientSecret**

строка

Обязательный параметр

Пароль приложения, созданного в OIDC-провайдере.

- **spec.oidc.enableBasicAuth**

булевый

Включает возможность basic-авторизации для Kubernetes API server.

В качестве credentials для basic-авторизации указываются логин и пароль пользователя из приложения, созданного в OIDC (возможно включить при указании только одного провайдера с типом OIDC/Crowd).

Работает **только** при включенном параметре `publishAPI`.

Полученные от IdP данные авторизации и групп сохраняются в кеш на 10 секунд.

- **spec.oidc.getUserInfo**

булевый

Запрашивать дополнительные данные об успешно подключенном пользователе.

Подробнее...

По умолчанию: `false`

- **spec.oidc.insecureSkipEmailVerified**

булевый

Игнорировать информацию о статусе подтверждения email пользователя.

Как именно подтверждается email, решает сам провайдер. В ответе от провайдера приходит лишь информация — подтвержден email или нет.

По умолчанию: `false`

- **spec.oidc.insecureSkipVerify**

булевый

Не производить проверку подлинности провайдера с помощью TLS. Небезопасно, не рекомендуется использовать в production-окружениях.

По умолчанию: `false`

- **spec.oidc.issuer**

строка

Обязательный параметр

Адрес OIDC-провайдера.

Пример:

```
issuer: https://accounts.google.com
```

- **spec.oidc.promptType**

строка

Определяет — должен ли Issuer запрашивать подтверждение и давать подсказки при аутентификации.

По умолчанию будет запрошено подтверждение при первой аутентификации. Допустимые значения могут изменяться в зависимости от Issuer.

По умолчанию: `consent`

■ **spec.oidc.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

■ **spec.oidc.scopes**

массив строк

Список полей для включения в ответ при запросе токена.

Score — это одиночный токен, он не может содержать пробельные символы: протокол OAuth2 передаёт score'ы списком, разделённым пробелами.

По умолчанию: `["openid","profile","email","groups","offline_access"]`

■ **Элемент массива**

строка

Шаблон: `^\S+$`

■ **spec.oidc.userIDKey**

строка

Claim, который будет использован для получения ID пользователя.

По умолчанию: `sub`

■ **spec.oidc.userNameKey**

строка

Claim, который будет использован для получения имени пользователя.

По умолчанию: `name`

○ **spec.saml**

объект

Параметры провайдера SAML (можно указывать, только если `type: SAML`).

■ **spec.saml.allowedGroups**

массив строк

Если задано, вход будет разрешён только членам хотя бы одной из перечисленных групп.

■ **spec.saml.emailAttr**

строка

SAML-атрибут для маппинга в claim email.

По умолчанию: `email`

■ **spec.saml.entityIssuer**

строка

Значение Issuer, ожидаемое в SAML-ответе. По умолчанию — URL callback.

■ **spec.saml.filterGroups**

булевый

Если `true`, коннектор будет фильтровать claim groups, оставляя только группы из списка `allowedGroups`.

По умолчанию: `false`

■ **spec.saml.groupsAttr**

строка

SAML-атрибут для маппинга в claim groups.

■ **spec.saml.groupsDelim**

строка

Разделитель для разбиения значения атрибута groups, если оно является одной строкой.

■ **spec.saml.insecureSkipSignatureValidation**

булевый

Пропустить проверку подписи SAML-ответов. Небезопасно, использовать только для тестирования.

По умолчанию: `false`■ **spec.saml.nameIDPolicyFormat**

строка

Формат NameIDPolicy для запроса к IdP.

Допустимые значения: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`■ **spec.saml.redirectURI**

строка

URI обратного вызова.

Переопределяет URL callback по умолчанию. Должен быть установлен в `https://<dex-domain>/callback`.■ **spec.saml.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

■ **spec.saml.ssoIssuer**

строка

Значение Issuer, ожидаемое в SAML-ответе от IdP.

■ **spec.saml.ssoURL**

строка

Обязательный параметр

URL для отправки SAML AuthnRequest (SSO URL провайдера).

Пример:

```
ssoURL: https://idp.example.com/saml/sso
```

■ **spec.saml.usernameAttr**

строка

SAML-атрибут для маппинга в claim username.

По умолчанию: `name`○ **spec.type**

строка

Обязательный параметр

Тип внешнего провайдера.

Допустимые значения: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описывает конфигурацию подключения стороннего провайдера.

С его помощью можно гибко настроить интеграцию каталога учетных записей с Kubernetes.

Пример использования...

- **spec**
объект

Обязательный параметр

- **spec.bitbucketCloud**
объект

Параметры провайдера Bitbucket Cloud (можно указывать, только если `type: BitbucketCloud`).

- **spec.bitbucketCloud.clientID**
строка

Обязательный параметр

ID приложения, созданного в Bitbucket Cloud (Key).

- **spec.bitbucketCloud.clientSecret**
строка

Обязательный параметр

Secret приложения, созданного в Bitbucket Cloud (Secret).

- **spec.bitbucketCloud.includeTeamGroups**
булевый

Включает в список команд все группы команды, в которых состоит пользователь.

Пример групп пользователя с включенной опцией:

```
groups=["my_team", "my_team/administrators", "my_team/members"]
```

По умолчанию: `false`

- **spec.bitbucketCloud.teams**
массив строк

Список-фильтр команд, допустимых для приема из Bitbucket Cloud.

Токен пользователя будет содержать пересечение множеств команд из Bitbucket Cloud и команд из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Токен будет содержать команды пользователя в claim `groups`, как и у других провайдеров.

- **spec.crowd**
объект

Параметры провайдера Crowd (можно указывать, только если `type: Crowd`).

- **spec.crowd.baseURL**
строка

Обязательный параметр

Адрес Crowd.

Пример:

```
baseURL: https://crowd.example.com/crowd
```

- **spec.crowd.clientID**
строка

Обязательный параметр

ID приложения, созданного в Crowd (Application Name).

- **spec.crowd.clientSecret**
строка

Обязательный параметр

Пароль приложения, созданного в Crowd (Password).

- **spec.crowd.enableBasicAuth**
булевый

Включает возможность базовой аутентификации (Basic Authentication) для Kubernetes API server.

В качестве учетных данных используются логин и пароль пользователя Crowd (из приложения, настроенного для этого провайдера).

Ограничения:

- работает только при включенном `publishAPI`;
- может быть включено только для одного провайдера аутентификации типа OIDC или Crowd в кластере.

Данные аутентификации и группы, полученные от провайдера аутентификации, кешируются на 10 секунд.

- **spec.crowd.groups**
массив строк

Список-фильтр групп, допустимых для приема из Crowd.

Токен пользователя будет содержать пересечение множеств групп из Crowd и групп из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все группы из Crowd.

- **spec.crowd.usernamePrompt**
строка

Строка, которая будет отображаться возле поля для имени пользователя в форме ввода логина и пароля.

По умолчанию: `Crowd username`

- **spec.displayName**
строка

Обязательный параметр

Имя провайдера, которое будет отображено на странице выбора провайдера для аутентификации.

Если настроен всего один провайдер, страница выбора провайдера показываться не будет.

- **spec.enabled**
булевый

Если значение `false`, провайдер будет проигнорирован при формировании конфигурации Dex и не появится на странице логина.

По умолчанию: `true`

- **spec.github**
объект

Параметры провайдера GitHub (можно указывать? только если `type: Github`).

- **spec.github.clientID**
строка

Обязательный параметр

ID организации на GitHub.

- **spec.github.clientSecret**

строка

Обязательный параметр

Secret организации на GitHub.

- **spec.github.orgs**

массив объектов

Массив названий организаций в GitHub.

- **spec.github.orgs.name**

строка

Обязательный параметр

Название организации.

- **spec.github.orgs.teams**

массив строк

Список команд, допустимых для приема из GitHub.

Токен пользователя будет содержать объединенное множество команд из GitHub и команд из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все команды из GitHub.

- **spec.github.teamNameField**

строка

Формат команд, которые будут получены из GitHub.

Если в организации `асме` есть группа `Site Reliability Engineers`, в случае:

- `name` будет получена группа с именем `['асме:Site Reliability Engineers']`;
- `slug` будет получена группа с именем `['асме:site-reliability-engineers']`;
- `both` будут получены группы с именами `['асме:Site Reliability Engineers', 'асме:site-reliability-engineers']`.

По умолчанию: `name`

Допустимые значения: `name`, `slug`, `both`

- **spec.github.useLoginAsID**

булевый

Позволяет вместо использования внутреннего GitHub ID использовать имя пользователя.

- **spec.gitlab**

объект

Параметры провайдера GitLab (можно указывать, только если `type: Gitlab`).

- **spec.gitlab.baseURL**

строка

Адрес GitLab.

Пример:

```
baseURL: https://gitlab.example.com
```

- **spec.gitlab.clientID**

строка

Обязательный параметр

ID приложения, созданного в GitLab (Application ID).

- **spec.gitlab.clientSecret**

строка

Обязательный параметр

Secret приложения, созданного в GitLab (Secret).

- **spec.gitlab.groups**

массив строк

Список-фильтр групп (**пути групп — path**, а не имена), допустимых для приема из GitLab.

Токен пользователя будет содержать пересечение множеств групп из GitLab и групп из этого списка. Если множество окажется пустым, авторизация не будет считаться успешной.

Если параметр не указан, токен пользователя будет содержать все группы из GitLab.

- **spec.gitlab.rootCAData**

строка

A CA chain to validate the provider in PEM format.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.gitlab.useLoginAsID**

булевый

Позволяет вместо использования внутреннего GitLab ID, использовать имя пользователя.

- **spec.ldap**

объект

Параметры провайдера LDAP.

- **spec.ldap.bindDN**

строка

Путь до сервис-аккаунта приложения в LDAP.

Пример:

```
bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
```

- **spec.ldap.bindPW**

строка

Пароль для сервис-аккаунта приложения в LDAP.

Пример:

```
bindPW: password
```

- **spec.ldap.enableBasicAuth**

булевый

Включает возможность базовой аутентификации (Basic Authentication) для Kubernetes API server с использованием этого LDAP-провайдера.

В качестве учетных данных используются логин и пароль пользователя из LDAP. Эта функция недоступна если в кластере присутствует более одного провайдера аутентификации (IdP) типа `OIDC`, `Crowd` или `LDAP`.

Работает **только** при включенном параметре `publishAPI`.

Полученные от IdP данные аутентификации и групп сохраняются в кеш на 10 секунд.

- **spec.ldap.groupSearch**

объект

Настройки фильтра для поиска групп для указанного пользователя.

Подробнее о процессе фильтрации...

■ **spec.ldap.groupSearch.baseDN**

строка

Обязательный параметр

Откуда будет начат поиск групп

Пример:

```
baseDN: cn=users,dc=example,dc=com
```

■ **spec.ldap.groupSearch.filter**

строка

Фильтр для директории с группами.

Пример:

```
filter: "(objectClass=person)"
```

■ **spec.ldap.groupSearch.nameAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранится уникальное имя группы.

Пример:

```
nameAttr: name
```

■ **spec.ldap.groupSearch.userMatchers**

массив объектов

Обязательный параметр

Список сопоставлений атрибута имени юзера с именем группы.

■ **spec.ldap.groupSearch.userMatchers.groupAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранятся имена пользователей, состоящих в группе.

Пример:

```
groupAttr: member
```

■ **spec.ldap.groupSearch.userMatchers.userAttr**

строка

Обязательный параметр

Имя атрибута, в котором хранится имя пользователя.

Пример:

```
userAttr: uid
```

■ **spec.ldap.host**

строка

Обязательный параметр

Адрес и порт (опционально) LDAP-сервера.

Пример:


```
host: ldap.example.com:636
```

- **spec.ldap.insecureNoSSL**

булевый

Подключаться к каталогу LDAP не по защищенному порту.

По умолчанию: `false`

- **spec.ldap.insecureSkipVerify**

булевый

Не производить проверку подлинности провайдера с помощью TLS. Небезопасно, не рекомендуется использовать в production-окружениях.

По умолчанию: `false`

- **spec.ldap.kerberos**

объект

Опциональные настройки Kerberos (SPNEGO) SSO для LDAP.

- **spec.ldap.kerberos.enabled**

булевый

Включить поток аутентификации Kerberos SPNEGO для данного LDAP-провайдера.

По умолчанию: `false`

- **spec.ldap.kerberos.expectedRealm**

строка

Необязательный требуемый realm (без учета регистра). Если указан и не совпадает с realm билета — вход отклоняется.

- **spec.ldap.kerberos.fallbackToPassword**

булевый

Если `true`, при отсутствии/некорректном заголовке `Authorization: Negotiate` будет показана форма ввода логина/пароля вместо ответа 401 с `WWW-Authenticate: Negotiate`.По умолчанию: `false`

- **spec.ldap.kerberos.keytabSecretName**

строка

Имя секрета в неймспейсе `d8-user-authn`, содержащего ключ `krb5.keytab`, который монтируется в под Dex.

- **spec.ldap.kerberos.krb5ConfConfigMapName**

строка

Необязательное имя ConfigMap с ключом `krb5.conf` для монтирования в `/etc/krb5.conf` (обычно не требуется для офлайн-проверки билетов).

- **spec.ldap.kerberos.krb5ConfSecretName**

строка

Необязательное имя секрета с ключом `krb5.conf` для монтирования в `/etc/krb5.conf` (обычно не требуется для офлайн-проверки билетов).

- **spec.ldap.kerberos.usernameFromPrincipal**

строка

Способ сопоставления Kerberos principal с именем пользователя в LDAP: `localpart`, `sAMAccountName` или `userPrincipalName`.По умолчанию: `Localpart`Допустимые значения: `Localpart`, `SAMAccountName`, `UserPrincipalName`

- **spec.ldap.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

■ **spec.ldap.startTLS**

булевый

Использовать STARTTLS для шифрования.

По умолчанию: `false`

■ **spec.ldap.userSearch**

объект

Обязательный параметр

Настройки фильтров пользователей, которые помогают сначала отфильтровать директории, в которых будет производиться поиск пользователей, а затем найти пользователя по полям (его имени, адресу электронной почты или отображаемому имени).

Подробнее о процессе фильтрации...

■ **spec.ldap.userSearch.baseDN**

строка

Обязательный параметр

Откуда будет начат поиск пользователей.

Пример:

```
baseDN: cn=users,dc=example,dc=com
```

■ **spec.ldap.userSearch.emailAttr**

строка

Обязательный параметр

Имя атрибута, из которого будет получен email пользователя.

Пример:

```
emailAttr: mail
```

■ **spec.ldap.userSearch.filter**

строка

Позволяет добавить фильтр для директории с пользователями.

Пример:

```
filter: "(objectClass=person)"
```

■ **spec.ldap.userSearch.idAttr**

строка

Обязательный параметр

Имя атрибута, из которого будет получен ID пользователя.

Пример:

```
idAttr: uid
```

■ **spec.ldap.userSearch.nameAttr**

строка

Атрибут отображаемого имени пользователя.

Пример:

```
nameAttr: name
```

- **spec.ldap.userSearch.username**

строка

Обязательный параметр

Имя атрибута, из которого будет получен username пользователя.

Пример:

```
username: uid
```

- **spec.ldap.usernamePrompt**

строка

Строка, которая будет отображаться возле поля для имени пользователя в форме ввода логина и пароля.

По умолчанию: LDAP username

Пример:

```
usernamePrompt: SSO Username
```

- **spec.oidc**

объект

Параметры провайдера OIDC (можно указывать, только если `type: OIDC`).

- **spec.oidc.allowedGroups**

массив строк

Список групп, которые должны быть у пользователя для успешной аутентификации.

Если список содержит группу, которой у пользователя нет, аутентификация будет отклонена.

Если параметр не задан, проверка принадлежности к группам не выполняется.

- **spec.oidc.basicAuthUnsupported**

булевый

Использовать POST-запросы для общения с провайдером вместо добавления токена в Basic Authorization header.

В большинстве случаев Dex сам определяет, какой запрос ему нужно сделать, но иногда включение этого параметра может помочь.

По умолчанию: false

- **spec.oidc.claimMapping**

объект

Некоторые провайдеры возвращают нестандартные claim'ы (например, mail). Claim mappings помогают Dex преобразовать их в стандартные claim'ы OIDC.

Dex может преобразовать нестандартный claim в стандартный, только если id_token, полученный от OIDC-провайдера, не содержит аналогичный стандартный claim.

- **spec.oidc.claimMapping.email**

строка

Claim, который будет использован для получения email пользователя.

По умолчанию: email

- **spec.oidc.claimMapping.groups**

строка

Claim, который будет использован для получения групп пользователя.

По умолчанию: `groups`

■ **spec.oidc.claimMapping.preferred_username**

строка

Claim, который будет использован для получения предпочтительного имени пользователя.

По умолчанию: `preferred_username`

■ **spec.oidc.claimMappingOverride**

булевый

Если включено, сопоставление claim'ов (`claimMapping`) будет переопределять стандартные claim'ы OIDC.

По умолчанию сопоставление claim'ов будет использоваться только в том случае, если стандартные claim'ы OIDC отсутствуют, например, если в `id_token` нет claim'a `email`, будет использоваться `claimMapping.email`.

По умолчанию: `false`

■ **spec.oidc.clientID**

строка

Обязательный параметр

ID приложения, созданного в OIDC-провайдере.

■ **spec.oidc.clientSecret**

строка

Обязательный параметр

Пароль приложения, созданного в OIDC-провайдере.

■ **spec.oidc.enableBasicAuth**

булевый

Включает возможность basic-авторизации для Kubernetes API server.

В качестве credentials для basic-авторизации указываются логин и пароль пользователя из приложения, созданного в OIDC (возможно включить при указании только одного провайдера с типом OIDC/Crowd).

Работает **только** при включенном параметре `publishAPI`.

Полученные от IdP данные авторизации и групп сохраняются в кеш на 10 секунд.

■ **spec.oidc.getUserInfo**

булевый

Запрашивать дополнительные данные об успешно подключенном пользователе.

Подробнее...

По умолчанию: `false`

■ **spec.oidc.insecureSkipEmailVerified**

булевый

Игнорировать информацию о статусе подтверждения email пользователя.

Как именно подтверждается email, решает сам провайдер. В ответе от провайдера приходит лишь информация — подтвержден email или нет.

По умолчанию: `false`

■ **spec.oidc.insecureSkipVerify**

булевый

Не производить проверку подлинности провайдера с помощью TLS. Небезопасно, не рекомендуется использовать в production-окружениях.

По умолчанию: `false`

■ **spec.oidc.issuer**

строка

Обязательный параметр

Адрес OIDC-провайдера.

Пример:

```
issuer: https://accounts.google.com
```

■ **spec.oidc.promptType**

строка

Определяет — должен ли Issuer запрашивать подтверждение и давать подсказки при аутентификации.

По умолчанию будет запрошено подтверждение при первой аутентификации. Допустимые значения могут изменяться в зависимости от Issuer.

По умолчанию: `consent`

■ **spec.oidc.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

■ **spec.oidc.scopes**

массив строк

Список полей для включения в ответ при запросе токена.

Score — это одиночный токен, он не может содержать пробельные символы: протокол OAuth2 передаёт score'ы списком, разделённым пробелами.

По умолчанию: `["openid", "profile", "email", "groups", "offline_access"]`

■ **Элемент массива**

строка

Шаблон: `^\S+$`

■ **spec.oidc.userIDKey**

строка

Claim, который будет использован для получения ID пользователя.

По умолчанию: `sub`

■ **spec.oidc.userNameKey**

строка

Claim, который будет использован для получения имени пользователя.

По умолчанию: `name`

○ **spec.saml**

объект

Параметры провайдера SAML (можно указывать, только если `type: SAML`).

■ **spec.saml.allowedGroups**

массив строк

Если задано, вход будет разрешён только членам хотя бы одной из перечисленных групп.

■ **spec.saml.emailAttr**

строка

SAML-атрибут для маппинга в claim email.

По умолчанию: `email`

- **spec.saml.entityIssuer**

строка

Значение Issuer, ожидаемое в SAML-ответе. По умолчанию — URL callback.

- **spec.saml.filterGroups**

булевый

Если `true`, коннектор будет фильтровать claim groups, оставляя только группы из списка `allowedGroups`.

По умолчанию: `false`

- **spec.saml.groupsAttr**

строка

SAML-атрибут для маппинга в claim groups.

- **spec.saml.groupsDelim**

строка

Разделитель для разбиения значения атрибута groups, если оно является одной строкой.

- **spec.saml.insecureSkipSignatureValidation**

булевый

Пропустить проверку подписи SAML-ответов. Небезопасно, использовать только для тестирования.

По умолчанию: `false`

- **spec.saml.nameIDPolicyFormat**

строка

Формат NameIDPolicy для запроса к IdP.

Допустимые значения: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`

- **spec.saml.redirectURI**

строка

URI обратного вызова.

Переопределяет URL callback по умолчанию. Должен быть установлен в `https://<dex-domain>/callback`.

- **spec.saml.rootCAData**

строка

Цепочка CA в формате PEM, используемая для валидации TLS.

Пример:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.saml.ssoIssuer**

строка

Значение Issuer, ожидаемое в SAML-ответе от IdP.

- **spec.saml.ssoURL**

строка

Обязательный параметр

URL для отправки SAML AuthnRequest (SSO URL провайдера).

Пример:

ssoURL: `https://idp.example.com/saml/sso`

- **spec.saml.usernameAttr**

строка

SAML-атрибут для маппинга в claim username.

По умолчанию: `name`

- **spec.type**

строка

Обязательный параметр

Тип внешнего провайдера.

Допустимые значения: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Group

Scope: Cluster

Version: v1alpha1

Содержит информацию о статических группах.

- **spec**

объект

Обязательный параметр

- **spec.members**

массив объектов

Обязательный параметр

Список членов группы.

- **spec.members.kind**

строка

Обязательный параметр

Тип члена группы.

Допустимые значения: `User`, `Group`

- **spec.members.name**

строка

Обязательный параметр

Имя пользователя (члена группы) или название группы (члена группы).

- **spec.name**

строка

Обязательный параметр

Уникальное имя группы.

- **spec.status**

объект

- **spec.status.errors**

массив объектов

- **spec.status.errors.message**

строка

- **spec.status.errors.objectRef**

объект

- **spec.status.errors.objectRef.kind**

строка

Допустимые значения: `User`, `Group`

- **spec.status.errors.objectRef.name**

строка

UserOperation

Scope: Cluster

- v1
- v1alpha1

Содержит информацию об операции над пользователем (сброс пароля, сброс 2FA, блокировка/разблокировка).

- **spec**
объект

Обязательный параметр

- **spec.initiatorType**
строка

Указывает, кем инициирована операция. Возможные значения:

- `admin` — администратором;
- `system` — автоматически системой;
- `self` — самим пользователем.

- **spec.lock**
объект

Параметры операции по блокировке пользователя.

- **spec.lock.for**
строка

Длительность блокировки.

Указывается в формате: `30s`, `10m`, `1h`, `2h30m`.

- **spec.resetPassword**
объект

Параметры операции по сбросу пароля.

- **spec.resetPassword.newPasswordHash**
строка

Новый пароль в виде bcrypt-хеша.

Хеш должен быть получен заранее и передан в поле CR.

- **spec.type**
строка

Обязательный параметр

Тип операции над пользователем. Возможные значения:

- `ResetPassword` — сброс пароля;
- `Reset2FA` — сброс двухфакторной аутентификации;
- `Lock` — блокировка пользователя;
- `Unlock` — разблокировка пользователя.

- **spec.user**
строка

Обязательный параметр

Имя пользователя (ссылка на ресурс User).

Пример:

```
user: jane.doe
```

- **status**
объект

- **status.completedAt**
строка

Время завершения операции.

- **status.message**
строка

Дополнительное сообщение о результате выполнения операции или причине ошибки.

- **status.phase**

строка

Текущий статус выполнения операции. Возможные значения:

- `Succeeded` — операция успешно выполнена;
- `Failed` — операция завершилась ошибкой.

Допустимые значения: `Succeeded`, `Failed`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Содержит информацию об операции над пользователем (сброс пароля, сброс 2FA, блокировка/разблокировка).

- **spec**

объект

Обязательный параметр

- **spec.initiatorType**

строка

Указывает, кем инициирована операция. Возможные значения:

- `admin` — администратором;
- `system` — автоматически системой;
- `self` — самим пользователем.

- **spec.lock**

объект

Параметры операции по блокировке пользователя.

- **spec.lock.for**

строка

Длительность блокировки.

Указывается в формате: `30s`, `10m`, `1h`, `2h30m`.

- **spec.resetPassword**

объект

Параметры операции по сбросу пароля.

- **spec.resetPassword.newPasswordHash**

строка

Новый пароль в виде bcrypt-хеша.

Хеш должен быть получен заранее и передан в поле CR.

- **spec.type**

строка

Обязательный параметр

Тип операции над пользователем. Возможные значения:

- `ResetPassword` — сброс пароля;
- `Reset2FA` — сброс двухфакторной аутентификации;
- `Lock` — блокировка пользователя;
- `Unlock` — разблокировка пользователя.

- **spec.user**

строка

Обязательный параметр

Имя пользователя (ссылка на ресурс User).

Пример:

```
user: jane.doe
```

- **status**

объект

- **status.completedAt**

строка

Время завершения операции

время завершения операции.

- **status.message**
строка

Дополнительное сообщение о результате выполнения операции или причине ошибки.

- **status.phase**
строка

Текущий статус выполнения операции. Возможные значения:

- `Succeeded` — операция успешно выполнена;
- `Failed` — операция завершилась ошибкой.

Допустимые значения: `Succeeded`, `Failed`

User

Scope: Cluster

- `v1`
- `v1alpha1`

Содержит информацию о статическом пользователе.

Пример использования...

- **spec**
объект

Обязательный параметр

- **spec.email**
строка

Обязательный параметр

Email пользователя.

Важно! При использовании совместно с модулем `user-authz`, для выдачи прав конкретному пользователю в качестве имени пользователя в CR `ClusterAuthorizationRule` необходимо указывать `email`.

Пример:

```
email: user@domain.com
```

- **spec.groups**
Параметр устарел
массив строк

Список групп, в которых у пользователя есть членство.

Т.к. параметр устарел, добавление пользователя в группы теперь осуществляется через ресурс `Group`.

- **spec.password**
строка

Обязательный параметр

Хеш пароля пользователя в явном виде или закодированный в Base64.

Для получения хеша пароля в Base64 можно воспользоваться командой `echo -n '<PASSWORD>' | httpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. Также можно воспользоваться онлайн-сервисом (например, <https://bcrypt-generator.com/>).

Пример:

```
password:
JDJ5JDEwJGRNWGVGVBUbkdUdYYVMYwDFPcGdZdk9HSy81LkdsNm5sdU9mUkhNwLQdDhuSlh6Szh
peS5H
```

- **spec.ttl**

строка

Время жизни учетной записи пользователя (TTL).

Задаётся в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

Указать TTL можно только 1 раз. При повторном изменении TTL, дата `expireAt` не обновляется.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

Пример:

```
ttl: 24h
```

- **spec.userID**

Параметр устарел
строка

Уникальное имя (ID) пользователя.

Больше не используется. Заполняется автоматически.

- **status**

объект

- **status.expireAt**

строка

Дата окончания действия учетной записи пользователя.

- Появляется только при заполнении поля `.spec.ttl`.
- При достижении этой даты учетная запись будет удалена.
- Синхронизируется раз в 5 минут. Возможен временной лаг между датой в этом поле и датой фактического удаления пользователя.

- **status.groups**

массив строк

Список групп, в которых у пользователя есть членство.

- **status.lock**

объект

Информация о блокировке учётной записи пользователя. Если объект присутствует и `state: true`, пользователь не может аутентифицироваться до указанного времени или пока блокировка не будет снята администратором.

Административные действия с учётной записью (блокировка, разблокировка, принудительная смена пароля, сброс 2FA) выполняются через ресурс `UserOperation`. Самостоятельный сброс пароля пользователем также выполняется через `UserOperation` (`initiatorType: self`). При блокировке учётной записи администратором или при принудительной смене пароля все активные сессии пользователя (объекты `Dex OfflineSessions` и `RefreshToken`) удаляются — пользователь обязан пройти повторную аутентификацию; в случае принудительной смены пароля при следующем успешном входе ему будет показана форма смены пароля.

- **status.lock.message**

строка

Человекочитаемое описание причины блокировки.

- **status.lock.reason**

строка

Машиночитаемая причина блокировки учётной записи. Используется для проверки политик и автоматических обработчиков.

Допустимые значения: `PasswordPolicyLockout`, `LockedByAdministrator`

- **status.lock.state**

булевый

Показывает, заблокирован ли в данный момент пользователь.

- `true` — учётная запись заблокирована.
- `false` — учётная запись активна.

- **status.lock.until**

строка

Время (в формате RFC3339), до которого учётная запись будет оставаться заблокированной. После наступления этого времени блокировка снимается автоматически.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Содержит информацию о статическом пользователе.

Пример использования...

- **список**
объект

Обязательный параметр

- **список.email**
строка

Обязательный параметр

Email пользователя. Должен быть в нижнем регистре.

Важно! При использовании совместно с модулем user-authz для выдачи прав конкретному пользователю в качестве имени пользователя в custom resource ClusterAuthorizationRule необходимо указывать `email`.

Важно! Email-адреса проверяются на уникальность без учёта регистра. Dex автоматически преобразует email-адреса в нижний регистр при аутентификации. Для новых пользователей заглавные буквы в email-адресах не допускаются. Существующие пользователи с заглавными буквами в email могут быть обновлены без изменения поля email.

Пример:

```
email: user@domain.com
```

- **список.groups**
Параметр устарел
массив строк

Список групп, в которых у пользователя есть членство.

Т.к. параметр устарел, добавление пользователя в группы теперь осуществляется через ресурс Group.

- **список.password**
строка

Обязательный параметр

Хеш пароля пользователя в явном виде или закодированный в Base64.

Для получения хеша пароля в Base64 можно воспользоваться командой `echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. Также можно воспользоваться онлайн-сервисом (например, <https://bcrypt-generator.com/>).

i

Важно. Если используется политика паролей с контролем сложности выше `None`, пользователь будет обязан сменить установленный таким образом пароль после первого успешного входа в систему.

Пример:

```
password:
JDJ5JDEwJGRNWGVGVbkdUdYYVMYwDFPcGdZdk9HSy81LkdsNm5sdU9mUkhNwLQdDhuSlh6Szh
peS5H
```

- **список.ttl**
строка

Время жизни учетной записи пользователя (TTL).

Задается в виде строки с указанием часов и минут: 30m, 1h, 2h30m, 24h.

Указать TTL можно только 1 раз. При повторном изменении TTL дата `expireAt` не обновляется.

Шаблон: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

Пример:

```
ttl: 24h
```

- **spec.userID**

строка

Уникальное имя (ID) пользователя.

Пример:

```
userID: '08a8684b-db88-4b73-90a9-3cd1661f5466'
```

- **status**

объект

- **status.expireAt**

строка

Дата окончания действия учетной записи пользователя:

- Появляется только при заполнении поля `.spec.ttl`.
- При достижении этой даты учетная запись будет удалена.
- Синхронизируется раз в 5 минут. Возможен временной лаг между датой в этом поле и датой фактического удаления пользователя.

- **status.groups**

массив строк

Список групп, в которых у пользователя есть членство.

- **status.lock**

объект

Информация о блокировке учётной записи пользователя. Если объект присутствует и `state: true`, пользователь не может аутентифицироваться до указанного времени или пока блокировка не будет снята администратором.

Административные действия с учётной записью (блокировка, разблокировка, принудительная смена пароля, сброс 2FA) выполняются через ресурс `UserOperation`. Самостоятельный сброс пароля пользователем также выполняется через `UserOperation` (`initiatorType: self`). При блокировке учётной записи администратором или при принудительной смене пароля все активные сессии пользователя (объекты `Dex OfflineSessions` и `RefreshToken`) удаляются — пользователь обязан пройти повторную аутентификацию; в случае принудительной смены пароля при следующем успешном входе ему будет показана форма смены пароля.

- **status.lock.message**

строка

Человекочитаемое описание причины блокировки.

- **status.lock.reason**

строка

Машиночитаемая причина блокировки учётной записи. Используется для проверки политик и автоматических обработчиков.

Допустимые значения: `PasswordPolicyLockout`, `LockedByAdministrator`

- **status.lock.state**

булевый

Показывает, заблокирован ли в данный момент пользователь.

- `true` — учётная запись заблокирована.
- `false` — учётная запись активна.

- **status.lock.until**

строка

Время (в формате RFC3339), до которого учётная запись будет оставаться

заблокированной. После наступления этого времени блокировка снимается автоматически.

Примеры конфигурации

Пример конфигурации модуля

В примере представлена конфигурация модуля `user-authn` в Deckhouse Kubernetes Platform.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    kubeconfigGenerator:
      - id: direct
        masterURI: https://159.89.5.247:6443
        description: "Direct access to kubernetes API"
    publishAPI:
      enabled: true
```

Примеры настройки провайдера

Проверка подключения провайдера

На странице провайдера в веб-интерфейсе DKP доступно действие **Проверить подключение**. Оно создаёт ресурс `DexProviderCheck` и ждёт, пока хук запишет результат проверки в его `status`.

Проверка выполняется, чтобы убедиться в следующем:

- указанный `DexProvider` существует и включён;
- эндпоинт `Dex discovery` внутри кластера доступен;
- эндпоинт внешнего провайдера доступен.

Для OIDC-провайдеров дополнительно читаются `discovery`-документ и `JWKS`. Для LDAP-провайдеров проверяется доступность `TCP/TLS/StartTLS` и, если включён `Kerberos`, наличие секрета с `keytab`. Проверка не выполняет полный сценарий входа и не валидирует пароль тестового пользователя.

GitHub

В примере представлены настройки провайдера для интеграции с GitHub.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company GitHub
  # Опционально: временно отключить провайдер, не удаляя CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

В организации GitHub необходимо создать новое приложение.

Для этого выполните следующие шаги:

- перейдите в `Settings` -> `Developer settings` -> `OAuth Apps` -> `Register a new OAuth application` и в качестве `Authorization callback URL` укажите адрес `https://dex.`

```
<modules.publicDomainTemplate>/callback.
```

Полученные `Client ID` и `Client Secret` укажите в Custom Resource `DexProvider`.

Если организация GitHub находится под управлением клиента, перейдите в `Settings` -> `Applications` -> `Authorized OAuth Apps` -> `<name of created OAuth App>` и нажмите `Send Request` для подтверждения. Попросите клиента подтвердить запрос, который придет к нему на email.

GitLab

В примере представлены настройки провайдера для интеграции с GitLab.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated GitLab
  gitlab:
    baseUrl: https://gitlab.example.com
    clientId: plainstring
    clientSecret: plainstring
    groups:
      - administrators
      - users
```

В GitLab проекта необходимо создать новое приложение.

Для этого выполните следующие шаги:

- **self-hosted**: перейдите в `Admin area` -> `Application` -> `New application` и в качестве `Redirect URI (Callback URL)` укажите адрес `https://dex.<modules.publicDomainTemplate>/callback`, выберите scopes: `read_user`, `openid`;
- **cloud GitLab.com**: под главной учетной записью проекта перейдите в `User Settings` -> `Application` -> `New application` и в качестве `Redirect URI (Callback URL)` укажите адрес `https://dex.<modules.publicDomainTemplate>/callback`, выберите scopes: `read_user`, `openid`;
- (для GitLab версии 16 и выше) включить опцию `Trusted/Trusted applications are automatically authorized on GitLab OAuth flow` при создании приложения.

Полученные `Application ID` и `Secret` укажите в Custom Resource `DexProvider`.

Atlassian Crowd

В примере представлены настройки провайдера для интеграции с Atlassian Crowd.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseUrl: https://crowd.example.com/crowd
    clientId: plainstring
    clientSecret: plainstring
    enableBasicAuth: true
    groups:
      - administrators
      - users
```

В соответствующем проекте Atlassian Crowd необходимо создать новое `Generic`-приложение.

Для этого выполните следующие шаги:

- перейдите в `Applications` -> `Add application`.

Полученные `Application Name` и `Password` укажите в Custom Resource `DexProvider`.

Группы CROWD укажите в lowercase-формате для Custom Resource `DexProvider`.

Bitbucket Cloud

В примере представлены настройки провайдера для интеграции с Bitbucket.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
    teams:
      - administrators
      - users
```

Для настройки аутентификации необходимо в Bitbucket в меню команды создать нового OAuth consumer.

Для этого выполните следующие шаги:

- перейдите в `Settings` -> `OAuth consumers` -> `New application` и в качестве `Callback URL` укажите адрес `https://dex.<modules.publicDomainTemplate>/callback`, разрешите доступ для `Account: Read` и `Workspace membership: Read`.

Полученные `Key` и `Secret` укажите в Custom Resource `DexProvider`.

OIDC (OpenID Connect)

Аутентификация через OIDC-провайдера требует регистрации клиента (или создания приложения). Сделайте это по документации вашего провайдера (например, Okta, Keycloak, Gluu или Blitz).

Полученные в ходе выполнения инструкции `clientID` и `clientSecret` укажите в Custom Resource `DexProvider`.

Ниже можно ознакомиться с некоторыми примерами.

Keycloak

После выбора `realm` для настройки, добавления пользователя в Users и создания клиента в разделе Clients с включенной аутентификацией, которая необходима для генерации `clientSecret`, выполните следующие шаги:

- Создайте в разделе `Client scopes` `scope` с именем `groups`, и назначьте ему предопределённое сопоставление `groups` («Client scopes» → «Client scope details» → «Mappers» → «Add predefined mappers»).
- В созданном ранее клиенте добавьте данный `scope` во вкладке Client scopes («Clients» → «Client details» → «Client Scopes» → «Add client scope»).
- В полях «Valid redirect URIs», «Valid post logout redirect URIs» и «Web origins» конфигурации клиента укажите `https://dex.<publicDomainTemplate>/*`, где `publicDomainTemplate` - это указанный шаблон DNS-имен кластера в модуле `global`.

В примере представлены настройки провайдера для интеграции с Keycloak:


```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Используйте имя вашего realm
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups

```

Если в Keycloak не используется подтверждение учетных записей по email, для корректной работы с ним в качестве провайдера аутентификации внесите изменения в настройку `Client scopes` одним из следующих способов:

- Удалите сопоставление `Email verified` («Client Scopes» → «Email» → «Mappers»). Это необходимо для корректной обработки значения `true` в поле `insecureSkipEmailVerified` и правильной выдачи прав пользователям с неподтвержденным email.
- Если отредактировать или удалить сопоставление `Email verified` невозможно, создайте отдельный Client Scope с именем `email_dkp` (или любым другим) и добавьте в него два сопоставления:
 - `email`: «Client Scopes» → `email_dkp` → «Add mapper» → «From predefined mappers» → `email`;
 - `email verified`: «Client Scopes» → `email_dkp` → «Add mapper» → «By configuration» → «Hardcoded claim». Укажите следующие поля:
 - «Name»: `email verified`;
 - «Token Claim Name»: `emailVerified`;
 - «Claim value»: `true`;
 - «Claim JSON Type»: `boolean`.

После этого в клиенте, зарегистрированном для кластера DKP, в разделе «Clients» для `Client scopes` замените значение `email` на `email_dkp`.

В ресурсе `DexProvider` укажите параметр `insecureSkipEmailVerified: true` и в поле `.spec.oidc.scopes` замените название Client Scope на `email_dkp`, следуя примеру:

```

scopes:
  - openid
  - profile
  - email_dkp
  - groups

```

Okta

В примере представлены настройки провайдера для интеграции с Okta:

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientId: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true

```

Blitz Identity Provider

На стороне провайдера Blitz Identity Provider при регистрации приложения необходимо указать URL для перенаправления пользователя после авторизации. При использовании `DexProvider` необходимо указать `https://dex.<publicDomainTemplate>/`, где `publicDomainTemplate` - указанный в модуле `global` шаблон DNS-имен кластера.

В примере представлены настройки провайдера для интеграции с Blitz Identity Provider:

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim для получения групп пользователя, группы пользователя
настраиваются на стороне провайдера Blitz Identity Provider
    clientId: clientId
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipEmailVerified: true # Установить true, если нет необходимости в
проверке email пользователя
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
      - profile
      - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC

```

Чтобы корректно отработывал выход из приложений (происходил отзыв токена и требовалась повторная авторизация), нужно установить `login` в значении параметра `promptType`.

Для обеспечения гранулированного доступа пользователя к приложениям необходимо:

- добавить параметр `allowedUserGroups` в `ModuleConfig` нужного приложения;
- добавить группы к пользователю (наименования групп должны совпадать как на стороне Blitz, так и на стороне Deckhouse).

Пример для Prometheus:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access

```

LDAP

В примере представлены настройки провайдера для интеграции с Active Directory:

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    insecureSkipVerify: true

    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!

    usernamePrompt: Email Address

    enableBasicAuth: true

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
        - userAttr: DN
          groupAttr: member
      nameAttr: cn

```

Настройка базовой аутентификации

Чтобы включить доступ к Kubernetes API с использованием базовой аутентификации (Basic Authentication) по учетным записям LDAP:

1. Убедитесь, что в конфигурации модуля `user-authn` включен параметр `publishAPI`.
2. Установите параметр `enableBasicAuth: true` в ресурсе `DexProvider` для LDAP.



Внимание. В кластере может быть только один провайдер аутентификации с включенным параметром `enableBasicAuth`.

После настройки пользователи смогут обращаться к Kubernetes API с помощью `kubectl`, используя свой логин и пароль в LDAP.

Пример `kubeconfig` для пользователя:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Путь к CA сертификату или insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default
```

Kerberos (SPNEGO) SSO для LDAP

Dex поддерживает аутентификацию без отображения формы ввода логина/пароля, которая реализуется с помощью механизма Kerberos (SPNEGO) для LDAP-коннектора. При использовании этого механизма браузер, доверяющий хосту Dex, отправляет `Authorization: Negotiate ...`, Dex валидирует Kerberos-билет по `keytab`, пропускает форму ввода логина/пароля, сопоставляет `principal` с LDAP-именем, получает группы и завершает OIDC-поток.

Минимальный пример (расширение спецификации LDAP-провайдера):

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
        - userAttr: uid
          groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab # Секрет в неймспейсе `d8-user-authn` с
ключом 'krb5.keytab'.
      expectedRealm: EXAMPLE.COM # Опционально, проверка realm (без учёта
регистра).
      usernameFromPrincipal: sAMAccountName #
localpart|sAMAccountName|userPrincipalName
      fallbackToPassword: false # По умолчанию false; если true – при
отсутствии/ошибке заголовка `Authorization: Negotiate` будет показана форма ввода
логина/пароля.

```

Примечания:

- Секрет `dex-kerberos-keytab` должен находиться в неймспейсе `d8-user-authn` и содержать ключ `krb5.keytab`.
- Один под Dex может обслуживать несколько LDAP+Kerberos провайдеров. У каждого — свой `keytab`. `krb5.conf` не требуется (Dex проверяет билеты офлайн по `keytab`). Для настройки аутентификации заведите в LDAP read-only-пользователя (service account). Полученные путь до пользователя и пароль укажите в параметрах `bindDN` и `bindPW` кастомного ресурса `DexProvider`. В параметре `bindPW` укажите пароль в открытом виде (plain text). Стратегии с передачей хешированных паролей не предусмотрены. Если в LDAP настроен анонимный доступ на чтение, настройки можно не указывать.

SAML

В примере представлены настройки провайдера для интеграции с SAML 2.0 Identity Provider (например, AD FS, Okta, Keycloak).

```

apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Корпоративный SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent

```

Для настройки SAML Identity Provider:

1. Зарегистрируйте Dex как Service Provider (SP) у вашего провайдера идентификации со следующими параметрами:
 - **ACS URL (Assertion Consumer Service):** `https://dex.<modules.publicDomainTemplate>/callback`
 - **Entity ID:** `https://dex.<modules.publicDomainTemplate>/callback`
 - **Формат идентификатора имени:** `persistent` или `emailAddress`
2. Настройте сопоставление атрибутов в провайдере идентификации для отправки атрибутов `email`, `name` (имя пользователя) и `groups` в SAML assertion.
3. Экпортируйте сертификат подписи провайдера идентификации и укажите его в поле `rootCAData` ресурса `DexProvider`.

i SAML не поддерживает refresh tokens нативно. Dex кеширует identity пользователя из первичного SAML assertion и возвращает её при последующих запросах refresh. Время жизни сессии контролируется настройками `expiry.refreshTokens` в конфигурации модуля `user-authn`.

Настройка OAuth2-клиента в Dex для подключения приложения

Этот вариант настройки подходит приложениям, которые имеют возможность использовать OAuth2-аутентификацию самостоятельно, без помощи `oauth2-proxy`. Чтобы позволить подобным приложениям взаимодействовать с Dex, используется Custom Resource `DexClient`.

```

apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
spec:
  redirectURIs:
    - https://app.example.com/callback
    - https://app.example.com/callback-reserve
  allowedGroups:
    - Everyone
    - admins
  trustedPeers:
    - opendistro-sibling

```

После создания такого ресурса в Dex будет зарегистрирован клиент с идентификатором (**clientID**) `dex-client-myname@mynamespace`.

Пароль доступа к клиенту (**clientSecret**) сохранится в секрете:

```
apiVersion: v1
kind: Secret
metadata:
  name: dex-client-myname
  namespace: mynamespace
type: Opaque
data:
  clientSecret: c2VjcGV0
```

Предоставление приложению доступа к Kubernetes API

Ресурс DexClient или DexAuthenticator можно настроить для получения токенов, которые принимает API-сервер Kubernetes. Для этого установите на ресурсе одну из следующих аннотаций со значением "true":

- `dexclient.deckhouse.io/allow-access-to-kubernetes` — для DexClient;
- `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` — для DexAuthenticator.

Пример настройки ресурса DexClient:

```
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec:
  redirectURIs:
    - https://app.example.com/callback
```

Аннотация регистрирует клиента в качестве доверенного (trusted peer) для привилегированного OAuth2-клиента `kubernetes`, что позволяет ему запрашивать ID-токены, предназначенные для API-сервера. В таком токене имя пользователя определяется значением `claim email`, а группы — значением `claim groups`. В результате приложение обращается к API-серверу от имени прошедшего аутентификацию пользователя и с предоставленными ему правами.



Предоставляемый таким образом доступ действует на уровне всего кластера, хотя DexClient и DexAuthenticator являются namespaced-ресурсами. По этой причине добавить аннотацию или изменить её значение на `"true"` может только субъект с правами на изменение конфигурации модуля `user-authn` — например, пользователь с ролью `d8:manage:permission:module:user-authn:edit`. Прав на создание ресурсов DexClient или DexAuthenticator в отдельном неймспейсе недостаточно.

Добавление аннотации ограничено независимо от указанного значения, включая `"false"`. Это необходимо для совместимости с версиями DKP до 1.78, в которых доступ предоставляется при наличии аннотации независимо от её значения. Если доступ к API-серверу Kubernetes не требуется, не добавляйте аннотацию.

Ограничение не распространяется на ресурсы, у которых аннотация уже установлена. Пользователь с правами на изменение такого ресурса может продолжать изменять его, в том числе удалить аннотацию или изменить значение так, чтобы доступ был отключён.

Локальная аутентификация

Локальная аутентификация обеспечивает проверку и управление доступом пользователей с возможностью настройки парольной политики, поддержкой двухфакторной аутентификации (2FA) и управлением группами.

Реализация соответствует требованиям безопасности ФСТЭК и рекомендациям OWASP, обеспечивая надёжную защиту доступа к кластеру и приложениям без необходимости интеграции с внешними системами аутентификации.

Создание пользователя

Придумайте пароль и укажите его хеш-сумму, закодированную в base64, в поле `password`. Email-адрес должен быть в нижнем регистре.

Для вычисления хеш-суммы пароля воспользуйтесь командой:

```
echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo
```

i

Если команда `htpasswd` недоступна, установите соответствующий пакет:

- `apache2-utils` — для дистрибутивов, основанных на Debian;
- `httpd-tools` — для дистрибутивов, основанных на CentOS;
- `apache2-htpasswd` — для ALT Linux.

Также можно воспользоваться онлайн-сервисом.

Обратите внимание, что в приведенном примере указан `ttl`.

```
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  # echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' |
  base64 -w0; echo
  password:
  'JDJ5JDEwJGRNWGVGVVBkdUdYYVMYWFpCgdZdk9HSy81LkdsNm5sdU9mUkhNWLQdDhuSlh6SzhpeS5H'
  ttl: 24h
```

Правила авторизации предоставляют права пользователю на основе email из выданного токена. По этой причине нельзя создать ресурс `User`, у которого значение `spec.email` совпадает с субъектом типа `User` в существующем ресурсе `AuthorizationRule` или `ClusterAuthorizationRule`. Это предотвращает незаметное предоставление прав новому пользователю.

Если совпадение необходимо, например, если правило авторизации было создано заранее, укажите другой email или установите для пользователя аннотацию `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"`.

При сопоставлении email приводится к нижнему регистру, поскольку в таком виде он записывается в токен. Например, `Admin@Example.com` совпадает с субъектом `admin@example.com`.

Ограничение не распространяется на уже существующих пользователей, чей email совпадает с субъектами правил авторизации. Такие пользователи продолжают работать, а при обнаружении совпадения выводится предупреждение.

Учитывайте это ограничение при декларативном управлении пользователями и правилами авторизации. Если пользователь и соответствующее правило создаются одновременно, правило может быть создано раньше пользователя, в результате чего создание пользователя будет отклонено. Чтобы разрешить такое совпадение, добавьте аннотацию `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` в манифест `User`.

При удалении пользователя соответствующий субъект из правила авторизации не удаляется автоматически. Пока он остаётся в правиле, права продолжают предоставляться для указанного email, а система выводит предупреждение об отсутствии соответствующего пользователя. Если впоследствии будет создан пользователь с тем же email, он получит эти права. Если такое поведение не требуется, удалите соответствующий субъект из правила авторизации.

Операции над локальным пользователем

Операции сброса пароля, сброса 2FA и блокировки выполняются через ресурс `UserOperation`. В поле `initiatorType` указывается, кто инициировал операцию: администратор (`admin`), система (`system`) или сам пользователь (`self`).

Административные операции

Для административных действий над локальными пользователями используйте команды `d8 iam user`. Они создают ресурс `UserOperation` с `initiatorType: admin`, дожидаются выполнения операции и выводят результат.

При выполнении операций `ResetPassword`, `Reset2FA` и `Lock` удаляются объекты `Dex OfflineSessions` и `RefreshToken`, принадлежащие пользователю. Это завершает активные `offline-сессии` пользователя и требует повторной аутентификации.

Пример интерактивного сброса пароля:

```
d8 iam user reset-password admin
```

Пример сброса пароля с чтением нового пароля из `stdin`:

```
echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
```

Пример сброса пароля с автоматической генерацией нового пароля:

```
d8 iam user reset-password admin --generate-password
```

Если пароль уже захеширован, передайте `bcrypt`-хеш без кодирования в `Base64`:

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

Пример сброса 2FA:

```
d8 iam user reset2fa admin
```

Пример блокировки пользователя на 30 минут:

```
d8 iam user lock admin 30m
```

Разблокировка пользователя:

```
d8 iam user unlock admin
```

По умолчанию команды ожидают завершения операции. Чтобы только создать `UserOperation` и вывести его имя, используйте флаг `--wait=false`.

Сброс пароля пользователем

Локальный пользователь может самостоятельно сбросить свой пароль в интерфейсе аутентификации `DKP`. При этом создаётся ресурс `UserOperation` с типом `ResetPassword` и `initiatorType: self`.

Самостоятельный сброс пароля доступен только для локальных учётных записей (встроенный коннектор `Local`). Пользователи, которые входят через внешние провайдеры аутентификации, должны обращаться к администратору соответствующей системы.

При сбросе пароля новый пароль должен соответствовать парольной политике, а активные сессии пользователя завершаются — требуется повторная аутентификация.

Добавление пользователя в группу

Пользователи могут быть объединены в группы для управления правами доступа. Пример манифеста ресурса `Group` для группы:

```

apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin

```

Здесь `members` — список пользователей, которые входят в группу.

Имя группы записывается в выданный токен без изменений. При этом оно неотличимо от имени группы, полученного от внешнего провайдера аутентификации. Поэтому нельзя создать ресурс `Group`, если значение `spec.name` совпадает с субъектом типа `Group` в существующем ресурсе `AuthorizationRule` или `ClusterAuthorizationRule`. Это предотвращает незаметное предоставление прав участникам новой группы.

Если совпадение необходимо, переименуйте группу или установите на ней аннотацию `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"`.

В отличие от `email`, имя группы при сопоставлении не приводится к нижнему регистру. По этой причине имена, различающиеся только регистром букв, считаются разными.

Ограничение не распространяется на уже существующие группы, имена которых совпадают с субъектами правил авторизации. Такие группы продолжают работать, а при обнаружении совпадения выводится предупреждение.

Учитывайте это ограничение при декларативном управлении группами и правилами авторизации. Если группа и соответствующее правило создаются одновременно, правило может быть создано раньше группы, в результате чего создание группы будет отклонено. Чтобы разрешить такое совпадение, добавьте аннотацию `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` в манифест `Group`.

При удалении группы соответствующий субъект из правила авторизации не удаляется автоматически. Пока он остаётся в правиле, права продолжают предоставляться группе с указанным именем. Группа, впоследствии созданная с тем же именем, получит эти права. Если такое поведение не требуется, удалите соответствующий субъект из правила авторизации.

Парольная политика

Настройки парольной политики позволяют контролировать сложность пароля, ротацию и блокировку пользователей:

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"

```

Описание полей:

- `complexityLevel` — уровень сложности пароля;
- `passwordHistoryLimit` — число предыдущих паролей, которые хранит система, чтобы предотвратить их повторное использование;
- `lockout` — настройки блокировки при превышении лимита неудачных попыток входа:

- `lockout.maxAttempts` — лимит неудачных попыток;
 - `lockout.lockDuration` — длительность блокировки пользователя;
- `rotation` — настройки ротации паролей:
 - `rotation.interval` — период обязательной смены пароля.

Двухфакторная аутентификация (2FA)

2FA позволяет повысить уровень безопасности, требуя ввести код из приложения-аутентификатора TOTP (например, Google Authenticator) при входе.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    staticUsers2FA:
      enabled: true
      issuerName: "awesome-app"
```

Описание полей:

- `enabled` — включает или отключает 2FA для всех статических пользователей;
- `issuerName` — имя, которое будет отображаться в приложении-аутентификаторе при добавлении аккаунта.

i

После включения 2FA каждый пользователь должен пройти процесс регистрации в приложении-аутентификаторе при первом входе.

Выдача прав пользователю или группе

Для настройки прав доступа используются параметры кастомного ресурса `ClusterAuthorizationRule`.

FAQ

Как защитить мое приложение?

Чтобы включить аутентификацию через Dex для приложения, выполните следующие шаги:

1. Создайте ресурс `DexAuthenticator`.

При создании `DexAuthenticator` в кластере создаётся экземпляр `oauth2-proxy`, подключённый к Dex. В указанном неймспейсе будут созданы объекты `Deployment`, `Service`, `Ingress`, `Secret`.

Пример ресурса `DexAuthenticator`:

```

apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  # Префикс имени подов Dex authenticator.
  # Например, если префикс имени `app-name`, то поды Dex authenticator будут
  # вида `app-name-dex-authenticator-7f698684c8-c5cjb`.
  name: app-name
  # Неймспейс, в котором будет развернут Dex authenticator.
  namespace: app-ns
spec:
  # Домен вашего приложения. Запросы на него будут перенаправляться для
  # прохождения аутентификации в Dex.
  applicationDomain: "app-name.kube.my-domain.com"
  # Отправлять ли заголовок `Authorization: Bearer` приложению. Полезно в связке
  # с auth_request в NGINX.
  sendAuthorizationHeader: false
  # Имя секрета с SSL-сертификатом.
  applicationIngressCertificateSecretName: "ingress-tls"
  # Название Ingress-класса, которое будет использоваться в создаваемом для Dex
  # authenticator Ingress-ресурсе.
  applicationIngressClassName: "nginx"
  # Время, на протяжении которого пользовательская сессия будет считаться
  # активной.
  keepUsersLoggedInFor: "720h"
  # Список групп, пользователям которых разрешено проходить аутентификацию.
  allowedGroups:
    - everyone
    - admins
  # Список адресов и сетей, с которых разрешено проходить аутентификацию.
  whitelistSourceRanges:
    - 1.1.1.1/32
    - 192.168.0.0/24

```

2. Подключите приложение к Dex.

Для этого добавьте в Ingress-ресурс приложения следующие аннотации:

- `nginx.ingress.kubernetes.io/auth-signin`: `https://$host/dex-authenticator/sign_in`
- `nginx.ingress.kubernetes.io/auth-response-headers`: `X-Auth-Request-User,X-Auth-Request-Email`
- `nginx.ingress.kubernetes.io/auth-url`: `https://<SERVICE_NAME>.<NS>.svc.{C_DOMAIN}/dex-authenticator/auth`, где:
 - `SERVICE_NAME` — имя сервиса (Service) аутентификатора. Как правило, оно соответствует формату `<NAME>-dex-authenticator` (`<NAME>` — это `metadata.name` ресурса `DexAuthenticator`);
 - `NS` — значение параметра `metadata.namespace` ресурса `DexAuthenticator`;
 - `C_DOMAIN` — домен кластера (параметр `clusterDomain` ресурса `ClusterConfiguration`).



Важно: Если имя `DexAuthenticator` (`<NAME>`) слишком длинное, имя сервиса (Service) может быть сокращено. Чтобы найти корректное имя сервиса, воспользуйтесь следующей командой (укажите имя неймспейса и аутентификатора):

```
d8 k get service -n <NS> -l "deckhouse.io/dex-authenticator-for=<NAME>" -o
jsonpath='{.items[0].metadata.name}'
```

Пример аннотаций Ingress-ресурса приложения для подключения к Dex:

```

annotations:
  nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-
  authenticator/sign_in
  nginx.ingress.kubernetes.io/auth-url: https://app-name-dex-authenticator.app-
  ns.svc.cluster.local/dex-authenticator/auth
  nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-
  Request-Email

```



Ingress приложения должен иметь настроенный TLS. DexAuthenticator не поддерживает Ingress-ресурсы, работающие только по HTTP.

Настройка ограничений на основе CIDR

В DexAuthenticator нет встроенной системы управления разрешением аутентификации на основе IP-адреса пользователя. Вместо этого вы можете воспользоваться аннотациями для Ingress-ресурсов:

- Если нужно ограничить доступ по IP и оставить прохождение аутентификации в Dex, добавьте аннотацию с указанием разрешенных CIDR через запятую:

```
nginx.ingress.kubernetes.io/whitelist-source-range: 192.168.0.0/32,1.1.1.1
```

- Чтобы разрешить доступ без аутентификации в Dex для пользователей из указанных сетей, а для остальных оставить обязательную аутентификацию, добавьте аннотацию:

```
nginx.ingress.kubernetes.io/satisfy: "any"
```

Как работает аутентификация с помощью DexAuthenticator



DexAuthenticator работает только по HTTPS. Ingress-ресурсы, настроенные только на HTTP, не поддерживаются.

Аутентификационные cookie устанавливаются с атрибутом `Secure`, что означает их передачу только через зашифрованные HTTPS-соединения.

Убедитесь, что для Ingress вашего приложения настроен TLS, прежде чем интегрировать его с DexAuthenticator.

- Dex в большинстве случаев перенаправляет пользователя на страницу входа провайдера и ожидает, что пользователь будет перенаправлен на его `/callback` URL. Однако такие провайдеры, как LDAP или Atlassian Crowd, не поддерживают этот вариант. Вместо этого пользователь должен ввести логин и пароль в форму входа в Dex, и Dex сам проверит учётные данные, выполнив запрос к API провайдера.
- DexAuthenticator устанавливает cookie с полным токеном обновления (вместо выдачи тикета, как для ID-токена), потому что Redis не сохраняет данные на диск. Если по тикету в Redis не найден ID-токен, пользователь сможет запросить новый ID-токен, предоставив токен обновления из cookie.
- DexAuthenticator выставляет HTTP-заголовок `Authorization`, равный значению ID-токена из Redis. Это необязательно для сервисов вроде `upmeter`, так как права доступа к `upmeter` менее детальные. Для Kubernetes Dashboard это критичная функциональность: Dashboard передаёт ID-токен дальше для доступа к API Kubernetes.

Как сгенерировать kubernetesconfig для доступа к Kubernetes API?

`kubernetesconfig` для удалённого доступа к кластеру через `kubectl` можно сгенерировать в веб-интерфейсе `kubeconfigurator`.

Настройте параметр `publishAPI`:

- Откройте настройки модуля `user-authn` (создайте ресурс ModuleConfig `user-authn`, если

его нет):

```
d8 k edit mc user-authn
```

- Добавьте следующую секцию в блок `settings` и сохраните изменения:

```
publishAPI:
  enabled: true
```

Имя `kubeconfig` зарезервировано для веб-интерфейса генерации `kubeconfig`. URL зависит от параметра `publicDomainTemplate` (например, при шаблоне вида `%s.kube.my` веб-интерфейс генерации `kubeconfig` будет доступен по адресу `kubeconfig.kube.my`, при `%s-kube.company.my` — по адресу `kubeconfig-kube.company.my`).

Настройка kube-apiserver

С помощью функций модуля `control-plane-manager` DKP автоматически настраивает `kube-apiserver`, выставляя следующие флаги так, чтобы модули `dashboard` и `kubeconfig-generator` могли работать в кластере.

Аргументы `kube-apiserver`, которые будут настроены

- `--oidc-client-id=kubernetes`
- `--oidc-groups-claim=groups`
- `--oidc-issuer-url=https://dex.%addonsPublicDomainTemplate%/`
- `--oidc-username-claim=email`

При использовании самоподписанных сертификатов для Dex добавляется ещё один аргумент, а в под `apiserver` монтируется файл CA:

- `--oidc-ca-file=/etc/kubernetes/oidc-ca.crt`

Как работает подключение к Kubernetes API с помощью сгенерированного kubeconfig

1. До начала работы `kube-apiserver` необходимо запросить конфигурационный эндпоинт OIDC провайдера (в нашем случае — Dex), чтобы получить `issuer` и настройки JWKS-эндпоинта.
2. `Kubeconfig generator` сохраняет ID-токен и Refresh-токен в файл `kubeconfig`.
3. После получения запроса с ID-токеном `kube-apiserver` проверяет, что токен подписан провайдером, настроенным на первом шаге, с помощью ключей, полученных с JWKS-эндпоинта. Затем сравнивает значения `claim iss` и `aud` из токена со значениями из конфигурации.

Как сменить секрет OAuth2-клиента kubernetes?

Секрет привилегированного OAuth2-клиента `kubernetes` хранится в Secret `kubernetes-dex-client-app-secret` в неймспейсе `d8-user-authn`. То же значение используют OAuth2-клиенты `kubeconfig-generator`, `kubeconfig-publish-api` и `kubeconfig-<slug>`, а также компонент `basic-auth-proxu`, которому секрет передаётся с помощью параметра `--ldap-client-secret`.

Удаление Secret не приводит к смене секрета: пока значение сохраняется во внутренних параметрах модуля, хук восстановит Secret с прежним значением.

Чтобы сменить секрет, выполните следующие действия:

1. Если неймспейс `d8-user-authn` управляется с помощью GitOps-инструмента, исключите Secret `kubernetes-dex-client-app-secret` из синхронизации. Иначе GitOps-инструмент восстановит прежнее значение.
2. Очистите поле `secret`:

```
d8 k -n d8-user-authn patch secret kubernetes-dex-client-app-secret --type merge -p '{"data": {"secret": ""}}'
```

3. Перезапустите DKP, чтобы хук зарегистрировал пустое поле и сгенерировал новый секрет:

```
d8 k -n d8-system rollout restart deployment/deckhouse
```

4. Убедитесь, что значение секрета изменилось:

```
d8 k -n d8-user-authn get secret kubernetes-dex-client-app-secret -o
jsonpath='{.data.secret}'
```

Если значение не изменилось, повторите шаги 2 и 3. Модуль мог восстановить прежнее значение до перезапуска DKP.

После смены секрета конфигурация использующих его компонентов в кластере обновится автоматически, а их поды будут перезапущены.

Ранее загруженные файлы kubeconfig содержат прежний клиентский секрет и больше не смогут обновлять токены. Скачайте такие файлы заново. Уже выданные ID-токены продолжают действовать до истечения их срока действия, определяемого параметром `settings.idTokenTTL` (по умолчанию — 10 минут).

Как включить SSO по Kerberos (SPNEGO) для LDAP?

Если на стороне клиента настроено доменное SSO (браузер доверяет домену Dex), Dex может принимать Kerberos-билеты по заголовку `Authorization: Negotiate` и выполнять аутентификацию без отображения формы ввода логина/пароля.

Включение SSO по Kerberos (SPNEGO) для LDAP:

1. В инфраструктуре клиента должен быть задан SPN `HTTP/<fqdn-dex>` для сервисного аккаунта и сгенерирован `keytab`.
2. В кластере создайте секрет в неймспейсе `d8-user-authn` с ключом `krb5.keytab`.
3. В ресурсе `DexProvider` (тип `LDAP`) включите блок `spec.ldap.kerberos` и настройте в нём параметры:
 - `enabled: true`;
 - `keytabSecretName: <имя секрета>`;
 - опционально: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`.

Dex автоматически смонтирует `keytab` и начнёт принимать SPNEGO. `krb5.conf` на сервере не обязателен — билеты проверяются по `keytab`.

Как настроить базовую аутентификацию для доступа к Kubernetes API через LDAP?

1. Включите параметр `publishAPI` в конфигурации модуля `user-authn`.
2. Создайте ресурс `DexProvider` типа `LDAP` и установите параметр `enableBasicAuth: true`.
3. Настройте RBAC для групп, получаемых из LDAP.
4. Передайте пользователям `kubeconfig` с настроенными параметрами базовой аутентификации (логин и пароль LDAP).



В кластере может быть только один провайдер аутентификации со включенным параметром `enableBasicAuth`.

Подробный пример описан в разделе Примеры конфигурации.

Как Dex защищен от подбора логина и пароля?

Каждому пользователю разрешено не более 20 попыток входа. После исчерпания лимита одна дополнительная попытка добавляется каждые 6 секунд.

Модуль user-authz

Модуль отвечает за генерацию объектов ролевой модели доступа, основанной на базе стандартного механизма RBAC Kubernetes. Модуль создает набор кластерных ролей (`ClusterRole`), подходящий для большинства задач по управлению доступом пользователей и групп.



С версии Deckhouse Kubernetes Platform v1.64 в модуле реализована экспериментальная модель ролевого доступа.

Функциональность экспериментальной и текущей моделей ролевого доступа несовместимы. Автоматическая конвертация ресурсов невозможна.

Экспериментальная ролевая модель

В отличие от текущей ролевой модели DKP, экспериментальная ролевая модель не использует ресурсы `ClusterAuthorizationRule` и `AuthorizationRule`. Настройка прав доступа выполняется стандартным для RBAC Kubernetes способом: с помощью создания ресурсов `RoleBinding` или `ClusterRoleBinding`, с указанием в них одной из подготовленных модулем `user-authz` ролей.

Модуль создаёт специальные агрегированные кластерные роли (`ClusterRole`). Используя эти роли в `RoleBinding` или `ClusterRoleBinding` можно решать следующие задачи:

- Управлять доступом к модулям определённой подсистеме применения.

Например, чтобы дать возможность пользователю, выполняющему функции сетевого администратора, настраивать *сетевые* модули (например, `cni-cilium`, `ingress-nginx`, `istio` и т. д.), можно использовать в `ClusterRoleBinding` роль `d8:manage:networking:manager`.

- Управлять доступом к *пользовательским* ресурсам модулей в рамках пространства имён.

Например, использование роли `d8:use:role:manager` в `RoleBinding`, позволит удалять/создавать/редактировать ресурс `PodLoggingConfig` в пространстве имён, но не даст доступа к *cluster-wide*-ресурсам `ClusterLoggingConfig` и `ClusterLogDestination` модуля `log-shipper`, а также не даст возможность настраивать сам модуль `log-shipper`.

Роли, создаваемые модулем, делятся на два класса:

- Use-роли — для назначения прав пользователям (например, разработчикам приложений) **в конкретном пространстве имён**.
- Manage-роли — для назначения прав администраторам.

i

Особенности комбинированного доступа: совместное использование `RoleBinding` или `ClusterRoleBinding` с `ClusterAuthorizationRule` (CAR) и `AuthorizationRule` (AR) для одного и того же пользователя.

Если в кластере включён режим мультитенантности (параметр `enableMultiTenancy: true`), итоговые права пользователя — это **объединение** всех действующих для него источников прав:

- уровень доступа из CAR — только внутри неймспейсов, разрешённых его параметрами `limitNamespaces` или `namespaceSelector`;
- уровень доступа из `AuthorizationRule` — внутри неймспейса, в котором создан `AuthorizationRule`;
- права из обычных `RoleBinding` — внутри их неймспейсов;
- права из обычных `ClusterRoleBinding` (не созданных модулем `user-authz`) — во всём кластере.

Ограничения по неймспейсам в CAR ограничивают только права, выданные этим CAR. Они не отменяют права, выданные через `RoleBinding`, `ClusterRoleBinding` или `AuthorizationRule` в других неймспейсах; при этом уровень доступа CAR на эти неймспейсы не распространяется. Например, если у пользователя есть CAR с `accessLevel: Editor`, ограниченный неймспейсом `ns-a`, и `RoleBinding` с ролью `view` в неймспейсе `ns-b`, пользователь получит права `Editor` в `ns-a` и права только на чтение в `ns-b`: `RoleBinding` не блокируется ограничениями CAR, а уровень `Editor` из CAR не «протекает» в `ns-b`.

В более старых версиях DKP вебхук модуля `user-authz` отклонял все запросы в неймспейсы, не указанные в CAR пользователя, даже при наличии `RoleBinding` на эти неймспейсы. Это ограничение снято: `RoleBinding` и CAR можно использовать для одного пользователя совместно.

Use-роли



Use-роль можно использовать только в ресурсе `RoleBinding`.

Use-роли предназначены для назначения прав пользователю **в конкретном пространстве имён**. Под пользователями понимаются, например, разработчики, которые используют настроенный администратором кластер для развёртывания своих приложений. Таким пользователям не нужно управлять модулями DKP или кластером, но им нужно иметь возможность, например, создавать свои Ingress-ресурсы, настраивать аутентификацию приложений и сбор логов с приложений.

Use-роль определяет права на доступ к `namespaced`-ресурсам модулей и стандартным `namespaced`-ресурсам Kubernetes (`Pod`, `Deployment`, `Secret`, `ConfigMap` и т. п.).

Модуль создаёт следующие use-роли:

- `d8:use:role:viewer` — позволяет в конкретном пространстве имён просматривать стандартные ресурсы Kubernetes, кроме секретов и ресурсов RBAC, а также выполнять аутентификацию в кластере;
- `d8:use:role:user` — дополнительно к роли `d8:use:role:viewer` позволяет в конкретном пространстве имён просматривать секреты и ресурсы RBAC, подключаться к подам, удалять поды (но не создавать или изменять их), выполнять `kubectl port-forward` и `kubectl proxy`, изменять количество реплик контроллеров;
- `d8:use:role:manager` — дополнительно к роли `d8:use:role:user` позволяет в конкретном пространстве имён управлять ресурсами модулей (например, `Certificate`, `PodLoggingConfig` и т. п.) и стандартными `namespaced`-ресурсами Kubernetes (`Pod`, `ConfigMap`, `CronJob` и т. п.);
- `d8:use:role:admin` — дополнительно к роли `d8:use:role:manager` позволяет в конкретном пространстве имён управлять ресурсами `ResourceQuota`, `ServiceAccount`, `Role`, `RoleBinding`, `NetworkPolicy`.

Manage-роли



Manage-роль не даёт доступа к пространству имён пользовательских приложений.

Manage-роль определяет доступ только к системным пространствам имён (начинающимся с `d8-` или `kube-`), и только к тем из них, в которых работают модули соответствующей подсистемы роли.

Manage-роли предназначены для назначения прав на управление всей платформой или её частью (подсистемой), но не самими приложениями пользователей. С помощью `manage`-роли можно, например, дать возможность администратору безопасности управлять модулями, отвечающими за безопасность кластера. В таком случае администратор безопасности сможет настраивать аутентификацию, авторизацию, политики безопасности и другие соответствующие параметры.



Manage-роль ограничивает, к каким модулям и неймспейсам обращается субъект, но не ограничивает привилегии, которые он может получить через доступные ему модули. Это особенно важно учитывать для подсистемы `security`: право управлять аутентификацией и авторизацией равносильно полному контролю над кластером.

Субъект, который может управлять модулем `user-authn`, способен зарегистрировать провайдер идентификации или сбросить учётные данные любого локального пользователя. Субъект, который может управлять модулем `user-authz`, способен изменить правила авторизации. В обоих случаях он может получить идентичность с любыми привилегиями, включая администратора кластера, поэтому при планировании доступа считайте `manage`-роль подсистемы `security` равной роли администратора кластера.

Manage-роль определяет права на доступ:

- к `cluster-wide`-ресурсам Kubernetes;
- к управлению модулями DKP (ресурсы `moduleConfig`) в рамках подсистемы роли, или всеми модулями DKP для роли `d8:manage:all:*`;
- к управлению `cluster-wide`-ресурсами модулей DKP в рамках подсистемы роли или всеми ресурсами модулей DKP для роли `d8:manage:all:*`;
- к системным пространствам имён (начинающимся с `d8-` или `kube-`), в которых работают модули подсистемы роли, или ко всем системным пространствам имён для роли `d8:manage:all:*`.

Формат названия manage-роли — `d8:manage:<SUBSYSTEM>:<ACCESS_LEVEL>`, где:

- **SUBSYSTEM** — подсистема роли. Может быть либо одной из подсистем списка, либо `all` для доступа в рамках всех подсистем;
- **ACCESS_LEVEL** — уровень доступа.

Примеры manage-ролей:

- `d8:manage:all:viewer` — доступ на просмотр конфигурации всех модулей DKP (ресурсы `moduleConfig`), их `cluster-wide`-ресурсов, их `namespaced`-ресурсов и стандартных объектов Kubernetes (кроме секретов и ресурсов RBAC) во всех системных пространствах имён (начинающихся с `d8-` или `kube-`);
- `d8:manage:all:manager` — аналогично роли `d8:manage:all:viewer`, только доступ на уровне `admin`, т. е. просмотр/создание/изменение/удаление конфигурации всех модулей DKP (ресурсы `moduleConfig`), их `cluster-wide`-ресурсов, их `namespaced`-ресурсов и стандартных объектов Kubernetes во всех системных пространствах имён (начинающихся с `d8-` или `kube-`);
- `d8:manage:observability:viewer` — доступ на просмотр конфигурации модулей DKP (ресурсы `moduleConfig`) из подсистемы `observability`, их `cluster-wide`-ресурсов, их `namespaced`-ресурсов и стандартных объектов Kubernetes (кроме секретов и ресурсов RBAC) в системных пространствах имён `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, `kube-prometheus-pushgateway`.

Модуль предоставляет два уровня доступа для администратора:

- `viewer` — позволяет просматривать стандартные ресурсы Kubernetes, конфигурацию модулей (ресурсы `moduleConfig`), `cluster-wide`-ресурсы модулей и `namespaced`-ресурсы модулей в пространстве имен модуля;
- `manager` — дополнительно к роли `viewer` позволяет управлять стандартными ресурсами Kubernetes, конфигурацией модулей (ресурсы `moduleConfig`), `cluster-wide`-ресурсами модулей и `namespaced`-ресурсами модулей в пространстве имен модуля;

Подсистемы ролевой модели

Каждый модуль DKP принадлежит определённой подсистеме. Для каждой подсистемы существует набор ролей с разными уровнями доступа. Роли обновляются автоматически при включении или отключении модуля.

Например, для подсистемы `networking` существуют следующие manage-роли, которые можно использовать в `ClusterRoleBinding`:

- `d8:manage:networking:viewer`
- `d8:manage:networking:manager`

Область действия роли зависит от того, к какой подсистеме она принадлежит:

- Область действия ролей из подсистемы `all` — все системные (начинающиеся с `d8-` или `kube-`) неймспейсы кластера.
- Область действия ролей из других подсистем — неймспейсы, в которых работают модули подсистемы (подробнее — в таблице состава подсистем), а также все `cluster-wide` объекты модулей подсистемы.

Таблица состава подсистем ролевой модели.

Подсистема	Модули подсистемы	Неймспейсы, используемые модулями подсистемы
all	Все модули	Все неймспейсы
deckhouse	• deckhouse • multitenancy-manager	• d8-multitenancy-manager • d8-system

Подсистема	Модули подсистемы	Неймспейсы, используемые модулями подсистемы
infrastructure	• chrony • cloud-provider-aws • cloud-provider-azure • cloud-provider-dynamix • cloud-provider-gcp • cloud-provider-huaweicloud • cloud-provider-openstack • cloud-provider-vcd • cloud-provider-vmware • cloud-provider-yandex • cloud-provider-zvirt • descheduler • istio • keepalived • local-path-provisioner • loki • node-manager • terraform-manager • vertical-pod-autoscaler	• d8-chrony • d8-cloud-instance-manager • d8-cloud-provider-aws • d8-cloud-provider-azure • d8-cloud-provider-dynamix • d8-cloud-provider-gcp • d8-cloud-provider-huaweicloud • d8-cloud-provider-openstack • d8-cloud-provider-vcd • d8-cloud-provider-vmware • d8-cloud-provider-yandex • d8-cloud-provider-zvirt • d8-descheduler • d8-istio • d8-keepalived • d8-local-path-provisioner • d8-monitoring • d8-system • kube-system
kubernetes	• control-plane-manager • flow-schema • kube-dns • kube-proxy • priority-class	• d8-system • kube-system
networking	• cilium-hubble • cni-cilium • cni-flannel • cni-simple-bridge • flow-schema • ingress-nginx • istio • kube-dns • kube-proxy • metallb • network-gateway • network-policy-engine • node-local-dns • openvpn • static-routing-manager	• d8-cni-cilium • d8-cni-flannel • d8-cni-simple-bridge • d8-ingress-nginx • d8-istio • d8-metallb • d8-network-gateway • d8-openvpn • d8-static-routing-manager • d8-system • kube-system
observability	• dashboard • extended-monitoring • log-shipper • monitoring-applications • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-kubernetes-control-plane • monitoring-ping • okmeter • operator-prometheus • prometheus • prometheus-metrics-adapter • prometheus-pushgateway • upmeter	• d8-log-shipper • d8-monitoring • d8-okmeter • d8-operator-prometheus • d8-upmeter • kube-prometheus-pushgateway

Подсистема	Модули подсистемы	Неймспейсы, используемые модулями подсистемы
security	- admission-policy-engine - cert-manager - operator-trivy - runtime-audit-engine - secret-copier - user-authn - user-authz	- d8-admission-policy-engine - d8-cert-manager - d8-operator-trivy - d8-runtime-audit-engine - d8-user-authn

Текущая ролевая модель

Особенности:

- Модуль реализует role-based-подсистему сквозной авторизации, расширяя функционал стандартного механизма RBAC.
- Настройка прав доступа происходит с помощью ресурсов.
- Управление доступом к инструментам масштабирования (параметр `allowScale` ресурса `ClusterAuthorizationRule` или `AuthorizationRule`).
- Управление доступом к форвардингу портов (параметр `portForwarding` ресурса `ClusterAuthorizationRule` или `AuthorizationRule`).
- Управление списком разрешённых пространств имён в формате `labelSelector` (параметр `namespaceSelector` ресурса `ClusterAuthorizationRule`).

В модуле, кроме использования RBAC, можно использовать удобный набор высокоуровневых ролей:

- `User` — позволяет получать информацию обо всех объектах (включая доступ к журналам логов), но не позволяет заходить в контейнеры, читать секреты и выполнять `port-forward`;
- `PrivilegedUser` — то же самое, что и `User`, но позволяет заходить в контейнеры, читать секреты, а также удалять поды (что обеспечивает возможность перезагрузки);
- `Editor` — то же самое, что и `PrivilegedUser`, но предоставляет возможность создавать, изменять и удалять все объекты, которые обычно нужны для прикладных задач;
- `Admin` — то же самое, что и `Editor`, но позволяет удалять служебные объекты (производные ресурсы, например `ReplicaSet`, `certmanager.k8s.io/challenges` и `certmanager.k8s.io/orders`);
- `ClusterEditor` — то же самое, что и `Editor`, но позволяет управлять ограниченным набором `cluster-wide`-объектов, которые могут понадобиться для прикладных задач (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet` и т. д). Роль для работы оператора кластера;
- `ClusterAdmin` — то же самое, что и `ClusterEditor` + `Admin`, но позволяет управлять служебными `cluster-wide`-объектами (производные ресурсы, например `MachineSets`, `Machines`, `OpenstackInstanceClasses` и т. п., а также `ClusterAuthorizationRule`, `ClusterRoleBindings` и `ClusterRole`). Роль для работы администратора кластера. **Важно**, что `ClusterAdmin`, поскольку он уполномочен редактировать `ClusterRoleBindings`, может **сам себе расширить полномочия**;
- `SuperAdmin` — разрешены любые действия с любыми объектами, при этом ограничения `namespaceSelector` и `limitNamespaces` продолжают работать.



Режим multi-tenancy (авторизация по пространству имён) в данный момент реализован по временной схеме и **не гарантирует безопасность!**

В случае, если в `ClusterAuthorizationRule`-ресурсе используется `namespaceSelector`, параметры `limitNamespaces` и `allowAccessToSystemNamespaces` не учитываются.

Если вебхук, который реализовывает систему авторизации, по какой-то причине будет недоступен, опции `allowAccessToSystemNamespaces`, `namespaceSelector` и `limitNamespaces` в `custom resource` перестанут применяться и пользователи будут иметь доступ во все пространства имён. После восстановления доступности вебхука опции продолжают работать.

Список доступа для каждой роли модуля по умолчанию

Сокращения для verbs:

- read - `get`, `list`, `watch`
- read-write - `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, `update`
- write - `create`, `delete`, `deletecollection`, `patch`, `update`

Роль **User**:

```
read:
- apiextensions.k8s.io/customresourcedefinitions
- apps/daemonsets
- apps/deployments
- apps/replicasets
- apps/statefulsets
- autoscaling.k8s.io/verticalpodautoscalers
- autoscaling/horizontalpodautoscalers
- batch/cronjobs
- batch/jobs
- configmaps
- discovery.k8s.io/endpointslices
- endpoints
- events
- events.k8s.io/events
- extensions/daemonsets
- extensions/deployments
- extensions/ingresses
- extensions/replicasets
- extensions/replicationcontrollers
- gateway.networking.k8s.io/backendtlspolicies
- gateway.networking.k8s.io/gatewayclasses
- gateway.networking.k8s.io/gateways
- gateway.networking.k8s.io/grpcroutes
- gateway.networking.k8s.io/httproutes
- gateway.networking.k8s.io/listenersets
- gateway.networking.k8s.io/referencegrants
- gateway.networking.k8s.io/tcproutes
- gateway.networking.k8s.io/tlsroutes
- gateway.networking.k8s.io/udproutes
- limitranges
- metrics.k8s.io/nodes
- metrics.k8s.io/pods
- namespaces
- networking.k8s.io/ingresses
- networking.k8s.io/networkpolicies
- nodes
- persistentvolumeclaims
- persistentvolumes
- pods
- pods/log
- policy/poddisruptionbudgets
- rbac.authorization.k8s.io/rolebindings
- rbac.authorization.k8s.io/roles
- replicationcontrollers
- resourcequotas
- serviceaccounts
- services
- storage.k8s.io/storageclasses
```

Роль **PrivilegedUser** (включает все правила из роли **User**):

```
create:
- pods/eviction
create,get:
- pods/attach
- pods/exec
delete,deletecollection:
- pods
read:
- secrets
```

Роль **Editor** (включает все правила из роли **User**, **PrivilegedUser**):

```

read-write:
  - apps/deployments
  - apps/statefulsets
  - autoscaling.k8s.io/verticalpodautoscalers
  - autoscaling/horizontalpodautoscalers
  - batch/cronjobs
  - batch/jobs
  - configmaps
  - discovery.k8s.io/endpointslices
  - endpoints
  - extensions/deployments
  - extensions/ingresses
  - gateway.networking.k8s.io/backendtlspolicies
  - gateway.networking.k8s.io/gateways
  - gateway.networking.k8s.io/grpcroutes
  - gateway.networking.k8s.io/httproutes
  - gateway.networking.k8s.io/listenersets
  - gateway.networking.k8s.io/referencegrants
  - gateway.networking.k8s.io/tcproutes
  - gateway.networking.k8s.io/tlsroutes
  - gateway.networking.k8s.io/udproutes
  - networking.k8s.io/ingresses
  - persistentvolumeclaims
  - policy/poddisruptionbudgets
  - serviceaccounts
  - services
write:
  - secrets

```

Роль **Admin** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**):

```

create,patch,update:
  - pods
delete,deletecollection:
  - apps/replicasets
  - extensions/replicasets
read-write:
  - deckhouse.io/authorizationrules
  - rbac.authorization.k8s.io/rolebindings
  - rbac.authorization.k8s.io/roles

```

Роль **ClusterEditor** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**):

```

read:
  - rbac.authorization.k8s.io/clusterrolebindings
  - rbac.authorization.k8s.io/clusterroles
write:
  - apiextensions.k8s.io/customresourcedefinitions
  - apps/daemonsets
  - extensions/daemonsets
  - gateway.networking.k8s.io/gatewayclasses
  - storage.k8s.io/storageclasses

```

Роль **ClusterAdmin** (включает все правила из роли **User**, **PrivilegedUser**, **Editor**, **Admin**, **ClusterEditor**):

```

proxy:
  - nodes
read-write:
  - deckhouse.io/clusterauthorizationrules
  - namespaces
  - nodes/configz
  - nodes/healthz
  - nodes/log
  - nodes/metrics
  - nodes/pods
  - nodes/proxy
  - nodes/stats
write:
  - limitranges
  - networking.k8s.io/networkpolicies
  - rbac.authorization.k8s.io/clusterrolebindings
  - rbac.authorization.k8s.io/clusterroles
  - rbac.authorization.k8s.io/rolebindings
  - rbac.authorization.k8s.io/roles
  - resourcequotas

```

Вы можете получить дополнительный список правил доступа для роли модуля из кластера (существующие пользовательские правила и нестандартные правила из других модулей Deckhouse):

```
D8_ROLE_NAME=Editor
kubectl get clusterrole -A -o jsonpath="{range .items[?(@.metadata.annotations.user-authz\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

Настройки

i

Внимание. Мы категорически не рекомендуем создавать поды и ReplicaSet'ы — эти объекты являются второстепенными и должны создаваться из других контроллеров. Доступ к созданию и изменению подов и ReplicaSet'ов полностью отсутствует.

Внимание. Режим multi-tenancy (авторизация по пространству имён) в данный момент реализован по временной схеме и **не гарантирует безопасность!** Если вебхук, который реализовывает систему авторизации, по какой-то причине будет недоступен, авторизация по пространству имён (опции `allowAccessToSystemNamespaces`, `namespaceSelector` и `limitNamespaces` в `custom resource`) перестанет работать и пользователи получат доступы во все пространства имён. После восстановления доступности вебхука всё вернётся на свои места.

Вся настройка прав доступа происходит с помощью `custom resources`.

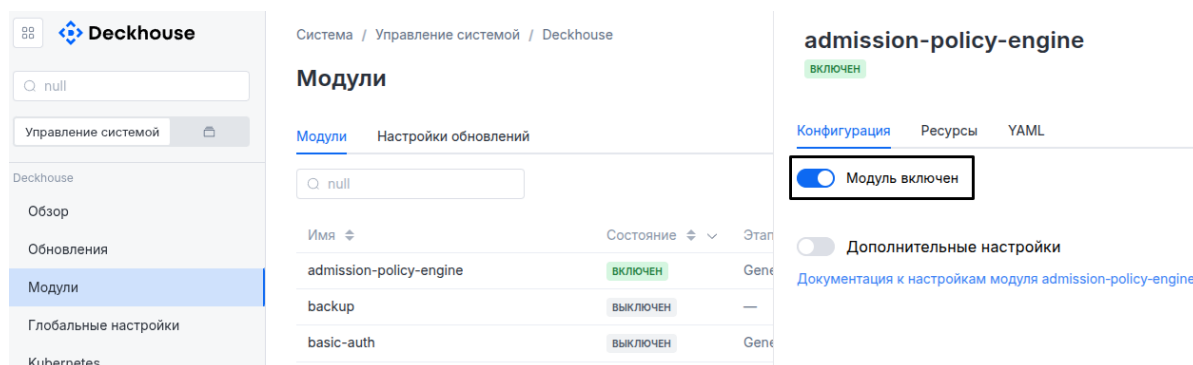
В модуле реализовано 2 алерта.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authz`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `user-authz`:

```
d8 system module enable user-authz
```

- **С помощью ModuleConfig user-authz.**

Установите `spec.enabled` в `true` или `false` в `ModuleConfig user-authz` (создайте его, при необходимости).

Пример манифеста для включения модуля `user-authz`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authz
spec:
  enabled: true
```

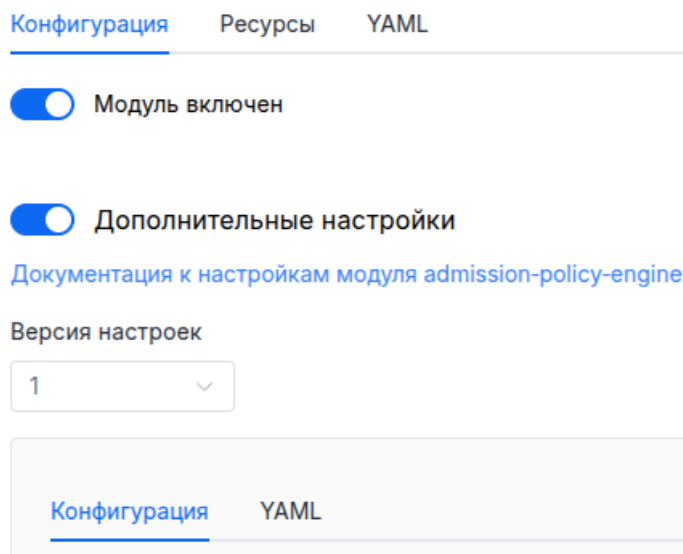
Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authz`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:



Вы также можете отредактировать объект `ModuleConfig user-authz` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `user-authz`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий `ModuleConfig user-authz` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc user-authz
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста `ModuleConfig user-authz`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста `ModuleConfig user-authz`:


```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authz
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.

```

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.controlPlaneConfigurator**

объект

Настройки параметров для модуля автоматической настройки kube-apiserver control-plane-manager.

- **settings.controlPlaneConfigurator.enabled**

булевый

Передавать ли в control-plane-manager параметры для настройки authz-webhook (см. параметры control-plane-manager'a).

При выключении этого параметра модуль control-plane-manager будет считать, что по умолчанию webhook-авторизация выключена, и, соответственно, если не будет дополнительных настроек, control-plane-manager будет стремиться вычеркнуть упоминания webhook-плагинов из манифеста. Даже если вы настроите манифест вручную.

По умолчанию: true

Примеры:

```
enabled: true
```

```
enabled: false
```

- **settings.enableMultiTenancy**

булевый

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Включить авторизацию по namespace (режим multi-tenancy).

Все настройки kube-apiserver, необходимые для работы авторизации по namespace, выполняются автоматически модулем control-plane-manager (подробнее).

По умолчанию: false

Примеры:

```
enableMultiTenancy: true
```

```
enableMultiTenancy: false
```

Custom Resources



Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

AuthorizationRule

Scope: Namespaced

Version: v1alpha1

Управляет настройками RBAC и авторизацией в пределах конкретного пространства имен (namespace'a).



Ресурс `AuthorizationRule` используется в текущей ролевой модели и не используется в экспериментальной ролевой модели.

- **spec**

объект

Обязательный параметр

- **spec.accessLevel**

строка

Уровень доступа:

- `User` — позволяет получать информацию обо всех объектах (включая доступ к журналам логов), но не позволяет заходить в контейнеры, читать `Secret`'ы и выполнять `port-forward`;
- `PrivilegedUser` — то же самое, что и `User`, но позволяет заходить в контейнеры, читать `Secret`'ы, а также удалять поды (что обеспечивает возможность перезагрузки);
- `Editor` — то же самое, что и `PrivilegedUser`, но предоставляет возможность создавать, изменять и удалять все объекты, которые обычно нужны для прикладных задач;
- `Admin` — то же самое, что и `Editor`, но позволяет удалять служебные объекты (производные ресурсы, например `ReplicaSet`, `certmanager.k8s.io/challenges` и `certmanager.k8s.io/orders`).

Допустимые значения: `User`, `PrivilegedUser`, `Editor`, `Admin`

Пример:

```
accessLevel: PrivilegedUser
```

- **spec.allowScale**

булевый

Разрешить/запретить масштабировать (выполнять `scale`) `Deployment`'ы и `StatefulSet`'ы.

По умолчанию: `false`

- **spec.portForwarding**

булевый

Разрешить/запретить выполнять `port-forward`.

По умолчанию: `false`

- **spec.subjects**

массив объектов

Обязательный параметр

Пользователи и/или группы, которым необходимо предоставить права.

Спецификация...

При использовании совместно с модулем `user-authn` обратите внимание на следующие нюансы:

- Для выдачи прав конкретному пользователю в качестве имени необходимо указывать его `email`.
- При указании группы убедитесь, что необходимые группы допускаются к получению от провайдера, то есть указаны в соответствующем `custom resource DexProvider`.
- **spec.subjects.kind**
строка

Обязательный параметр

Тип ресурса.

Допустимые значения: `User`, `Group`, `ServiceAccount`

Пример:

```
kind: Group
```

■ **spec.subjects.name**

строка

Обязательный параметр

Имя ресурса.

Пример:

```
name: some-group-name
```

■ **spec.subjects.namespace**

строка

Namespace для ServiceAccount.

Шаблон: `[a-z0-9]([a-z0-9]*[a-z0-9])?`

Длина: `1..63`

ClusterAuthorizationRule

Scope: Cluster

- `v1`
- `v1alpha1`

Управляет настройками RBAC и авторизацией по namespace.

Настройки определяют, какой уровень доступа назначен пользователю и/или группе.

- **spec**
объект

Обязательный параметр

- **spec.accessLevel**
строка

Уровень доступа:

- `User` — позволяет получать информацию обо всех объектах (включая доступ к журналам логов), но не позволяет заходить в контейнеры, читать Secret'ы и выполнять port-forward;
- `PrivilegedUser` — то же самое, что и `User`, но позволяет заходить в контейнеры, читать Secret'ы, а также удалять поды (что обеспечивает возможность перезагрузки);
- `Editor` — то же самое, что и `PrivilegedUser`, но предоставляет возможность создавать, изменять и удалять все объекты, которые обычно нужны для прикладных задач;
- `Admin` — то же самое, что и `Editor`, но позволяет удалять служебные объекты (производные ресурсы, например `ReplicaSet`, `certmanager.k8s.io/challenges` и `certmanager.k8s.io/orders`);
- `ClusterEditor` — то же самое, что и `Editor`, но позволяет управлять ограниченным набором cluster-wide-объектов, которые могут понадобиться для прикладных задач (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet` и т. д.). Роль для работы оператора кластера;
- `ClusterAdmin` — то же самое, что и `ClusterEditor` + `Admin`, но позволяет управлять служебными cluster-wide-объектами (производные ресурсы, например `MachineSets`, `Machines`, `OpenstackInstanceClasses` и т. п., а также `ClusterAuthorizationRule`, `ClusterRoleBindings` и `ClusterRole`). Роль для работы администратора кластера.

Важно! Так как `ClusterAdmin` уполномочен редактировать `ClusterRoleBindings`, он может сам себе расширить полномочия;

- **SuperAdmin** — разрешены любые действия с любыми объектами, при этом ограничения `namespaceSelector` и `limitNamespaces` (см. ниже) продолжат работать.

Допустимые значения: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

Пример:

```
accessLevel: PrivilegedUser
```

- **spec.additionalRoles**

массив объектов

Дополнительные роли, которые необходимо выдать для заданных `subjects`.

Параметр сделан на крайний случай, вместо него категорически рекомендуется использовать параметр `accessLevel`.

Пример:

```
additionalRoles:
- apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-write-all
- apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-read-all
```

- **spec.additionalRoles.apiGroup**

строка

Обязательный параметр

`apiGroup` для пользователей.

Пример:

```
apiGroup: rbac.authorization.k8s.io
```

- **spec.additionalRoles.kind**

строка

Обязательный параметр

Kind роли.

Допустимые значения: `ClusterRole`

Пример:

```
kind: ClusterRole
```

- **spec.additionalRoles.name**

строка

Обязательный параметр

Название роли.

Пример:

```
name: cluster-admin
```

- **spec.allowAccessToSystemNamespaces**

Параметр устарел

булевый

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Разрешить пользователю доступ в служебные namespace (`"kube-.*"`, `"d8-.*"`, `"loghouse"`, `"default"`).

Доступно только с включенным параметром `enableMultiTenancy`.

По умолчанию: `false`

- **spec.allowScale**
булевый

Разрешить/запретить масштабировать (выполнять scale) Deployment'ы и StatefulSet'ы.

По умолчанию: `false`

- **spec.limitNamespaces**
Параметр устарел
массив строк

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Список разрешенных namespaces в формате регулярных выражений.

Политика:

- Если список указан, разрешаем доступ только по нему.
- Если список не указан, считаем, что разрешено все, кроме системных namespaces (см. `spec.allowAccessToSystemNamespaces` ниже).

Доступно только с включенным параметром `enableMultiTenancy`.

Параметр устарел, используйте вместо него параметр `namespaceSelector`.

Пример:

```
limitNamespaces: production-.*
```

- **Элемент массива**
строка

- **spec.namespaceSelector**
объект

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Определяет доступные пространства имен на основе значений, указанных в параметре `namespaceSelector.labelSelector`.

Если параметр `namespaceSelector` указан, значения параметров `limitNamespaces` и `allowAccessToSystemNamespaces` **игнорируются**. Таким образом, будут доступны либо все пространства имен с лейблами, подпадающими под указанные в параметре `namespaceSelector.labelSelector` условия (включая системные пространства имен), либо все пространства имен (включая системные), если указан параметр `namespaceSelector.matchAny`.

Если параметр `namespaceSelector` не указан, набор доступных пространств имен определяется значением параметров `limitNamespaces` и `allowAccessToSystemNamespaces`. Если не указан ни один из параметров `namespaceSelector`, `limitNamespaces` и `allowAccessToSystemNamespaces`, будут доступны все пространства имен, за исключением системных (`kube-*`, `d8-*`, `loghouse`, `default`).

Доступно только с включенным параметром `enableMultiTenancy`.

- **spec.namespaceSelector.labelSelector**
объект

Фильтр по лейблам пространств имен.

Не может использоваться совместно с параметром `matchAny`.

Если указаны одновременно `matchExpressions` и `matchLabels`, учитываются оба фильтра (AND).

- **spec.namespaceSelector.labelSelector.matchExpressions**
массив объектов

Список фильтров на основе выражений.

Пример:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.namespaceSelector.labelSelector.matchExpressions.key**
строка

Имя лейбла.

- **spec.namespaceSelector.labelSelector.matchExpressions.operator**
строка

Оператор сравнения.

Допустимые значения: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.namespaceSelector.labelSelector.matchExpressions.values**
массив строк

Значение лейбла.

- **Элемент массива**
строка

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: `1..63`

- **spec.namespaceSelector.labelSelector.matchLabels**
объект

Фильтр на основе совпадения/несовпадения лейблов.

Пример:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.namespaceSelector.matchAny**
булевый

Разрешает доступ ко всем пространствам имен в кластере, включая системные.

Не может использоваться совместно с параметром `labelSelector`.

Так как с помощью параметра `labelSelector` можно предоставить доступ только к определенному перечню пространств имен, параметр `matchAny` необходим в случаях, когда нужно предоставить доступ ко всем пространствам имен.

Допустимые значения: `true`

- **spec.portForwarding**
булевый

Разрешить/запретить выполнять `port-forward`.

По умолчанию: `false`

- **spec.subjects**
массив объектов

Обязательный параметр

Пользователи и/или группы, которым необходимо предоставить права.

Спецификация...

При использовании совместно с модулем user-authn обратите внимание на следующие нюансы:

- Для выдачи прав конкретному пользователю в качестве имени необходимо указывать его `email`.
- При указании группы убедитесь, что необходимые группы допускаются к получению от провайдера, то есть указаны в соответствующем `custom resource DexProvider`.
- **spec.subjects.kind**
строка

Обязательный параметр

Тип ресурса.

Допустимые значения: `User`, `Group`, `ServiceAccount`

Пример:

```
kind: Group
```

- **spec.subjects.name**
строка

Обязательный параметр

Имя ресурса.

Пример:

```
name: some-group-name
```

- **spec.subjects.namespace**
строка

Namespace для ServiceAccount.

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: `1..63`

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Cluster-wide-ресурс для управления настройками RBAC и авторизацией.



Ресурс `ClusterAuthorizationRule` используется в текущей ролевой модели и не используется в экспериментальной ролевой модели.

- **spec**
объект

Обязательный параметр

- **spec.accessLevel**
строка

Уровень доступа:

- `User` — позволяет получать информацию обо всех объектах (включая доступ к журналам логов), но не позволяет заходить в контейнеры, читать Secret'ы и выполнять port-forward;
- `PrivilegedUser` — то же самое, что и `User`, но позволяет заходить в контейнеры, читать Secret'ы, а также удалять поды (что обеспечивает возможность перезагрузки);
- `Editor` — то же самое, что и `PrivilegedUser`, но предоставляет возможность создавать, изменять и удалять все объекты, которые обычно нужны для прикладных задач;
- `Admin` — то же самое, что и `Editor`, но позволяет удалять служебные объекты (производные ресурсы, например `ReplicaSet`, `certmanager.k8s.io/challenges` и `certmanager.k8s.io/orders`);
- `ClusterEditor` — то же самое, что и `Editor`, но позволяет управлять ограниченным набором cluster-wide-объектов, которые могут понадобиться для прикладных задач (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet` и т. д.).

Роль для работы оператора кластера;

- `ClusterAdmin` — то же самое, что и `ClusterEditor` + `Admin`, но позволяет управлять служебными cluster-wide-объектами (производные ресурсы, например `MachineSets`, `Machines`, `OpenstackInstanceClasses` и т. п., а также `ClusterAuthorizationRule`, `ClusterRoleBindings` и `ClusterRole`). Роль для работы администратора кластера.

Важно! Так как `ClusterAdmin` уполномочен редактировать `ClusterRoleBindings`, он может сам себе расширить полномочия;

- `SuperAdmin` — разрешены любые действия с любыми объектами, при этом ограничения `limitNamespaces` (см. ниже) продолжают работать.

Допустимые значения: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

Пример:

```
accessLevel: PrivilegedUser
```

о **spec.additionalRoles**

массив объектов

Дополнительные роли, которые необходимо выдать для заданных `subjects`.

Параметр сделан на крайний случай, вместо него категорически рекомендуется использовать параметр `accessLevel`.

Пример:

```
additionalRoles:
- apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-write-all
- apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-read-all
```

■ **spec.additionalRoles.apiGroup**

строка

Обязательный параметр

`apiGroup` для пользователей.

Пример:

```
apiGroup: rbac.authorization.k8s.io
```

■ **spec.additionalRoles.kind**

строка

Обязательный параметр

Kind роли.

Допустимые значения: `ClusterRole`

Пример:

```
kind: ClusterRole
```

■ **spec.additionalRoles.name**

строка

Обязательный параметр

Название роли.

Пример:


```
name: cluster-admin
```

- **spec.allowAccessToSystemNamespaces**

Параметр устарел
булевый

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Разрешить пользователю доступ в служебные namespace (["kube-.*", "d8-.*", "loghouse", "default"]).

Доступно только с включенным параметром enableMultiTenancy.

По умолчанию: `false`

- **spec.allowScale**

булевый

Разрешить/запретить масштабировать (выполнять scale) Deployment'ы и StatefulSet'ы.

По умолчанию: `false`

- **spec.limitNamespaces**

Параметр устарел
массив строк

Доступно в редакциях: BE, SE, SE+, EE, CSE Lite (1.73), CSE Pro (1.73)

Список разрешенных namespace в формате регулярных выражений.

Политика:

- Если список указан, разрешаем доступ только по нему.
- Если список не указан, считаем, что разрешено все, кроме системных namespace (см. `spec.allowAccessToSystemNamespaces` ниже).

Доступно только с включенным параметром enableMultiTenancy.

Параметр устарел, используйте вместо него параметр namespaceSelector (версия `v1` custom resource'a).

Пример:

```
limitNamespaces: production-.*
```

- **Элемент массива**
строка

- **spec.portForwarding**

булевый

Разрешить/запретить выполнять `port-forward`.

По умолчанию: `false`

- **spec.subjects**

массив объектов

Обязательный параметр

Пользователи и/или группы, которым необходимо предоставить права.

Спецификация...

При использовании совместно с модулем user-authn обратите внимание на следующие нюансы:

- Для выдачи прав конкретному пользователю в качестве имени необходимо указывать его `email`.
- При указании группы убедитесь, что необходимые группы допускаются к получению от провайдера, то есть указаны в соответствующем custom resource DexProvider.
- **spec.subjects.kind**
строка

Обязательный параметр

Тип ресурса.

Допустимые значения: `User`, `Group`, `ServiceAccount`

Пример:

```
kind: Group
```

■ **spec.subjects.name**

строка

Обязательный параметр

Имя ресурса.

Пример:

```
name: some-group-name
```

■ **spec.subjects.namespace**

строка

Namespace для ServiceAccount.

Шаблон: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

Длина: `1..63`

Примеры конфигурации

Пример назначения прав администратору кластера



Пример использует экспериментальную ролевую модель.

Для назначения прав администратору кластера используйте роль `d8:manage:all:manager` в `ClusterRoleBinding`.

Пример назначения прав администратору кластера (User `jane`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
```

Права, которые получит пользователь

Права, которые получит пользователь, будут ограничены рамками пространств имён, начинающихся с `d8-` или `kube-`.

Пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание ресурсов Kubernetes и модулей DKP.
- Изменение конфигурации модулей (просмотр, изменение, удаление и создание ресурсов `moduleConfig`).
- Выполнение следующих команд к подам и сервисам:
 - `kubectl attach`;
 - `kubectl exec`;
 - `kubectl port-forward`;
 - `kubectl proxy`.

Пример назначения прав сетевому администратору

i Пример использует экспериментальную ролевую модель.

Для назначения прав сетевому администратору на управление сетевой подсистемой кластера используйте роль `d8:manage:networking:manager` в `ClusterRoleBinding`.

Пример назначения прав сетевому администратору (User `jane`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:networking:manager
  apiGroup: rbac.authorization.k8s.io
```

Список прав, которые получит пользователь

Права, которые получит пользователь, будут ограничены следующим списком пространств имён модулей DKP из подсистемы `networking` (фактический список зависит от списка включённых в кластере модулей):

- `d8-cni-cilium`;
- `d8-cni-flannel`;
- `d8-cni-simple-bridge`;
- `d8-ingress-nginx`;
- `d8-istio`;
- `d8-metallb`;
- `d8-network-gateway`;
- `d8-openvpn`;
- `d8-static-routing-manager`;
- `d8-system`;
- `kube-system`.

Пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание *стандартных* ресурсов Kubernetes в пространстве имён модулей из подсистемы `networking`.

Пример ресурсов, которыми сможет управлять пользователь (список не полный):

- `Certificate`;
- `CertificateRequest`;
- `ConfigMap`;
- `ControllerRevision`;
- `CronJob`;
- `DaemonSet`;
- `Deployment`;
- `Event`;
- `HorizontalPodAutoscaler`;
- `Ingress`;
- `Issuer`;
- `Job`;
- `Lease`;
- `LimitRange`;
- `NetworkPolicy`;
- `PersistentVolumeClaim`;
- `Pod`;
- `PodDisruptionBudget`;
- `ReplicaSet`;
- `ReplicationController`;
- `ResourceQuota`;

- Role;
 - RoleBinding;
 - Secret;
 - Service;
 - ServiceAccount;
 - StatefulSet;
 - VerticalPodAutoscaler;
 - VolumeSnapshot.
- Просмотр, изменение, удаление и создание ресурсов в пространстве имён модулей из подсистемы `networking`.

Список ресурсов, которыми сможет управлять пользователь:

- EgressGateway;
 - EgressGatewayPolicy;
 - FlowSchema;
 - IngressClass;
 - IngressIstioController;
 - IngressNginxController;
 - IPRuleSet;
 - IstioFederation;
 - IstioMulticluster;
 - RoutingTable.
- Изменение конфигурации модулей (просмотр, изменение, удаление и создание ресурсов `moduleConfig`) из подсистемы `networking`.

Список модулей, которыми сможет управлять пользователь:

- cilium-hubble;
 - cni-cilium;
 - cni-flannel;
 - cni-simple-bridge;
 - flow-schema;
 - ingress-nginx;
 - istio;
 - kube-dns;
 - kube-proxy;
 - metallb;
 - network-gateway;
 - network-policy-engine;
 - node-local-dns;
 - openvpn;
 - static-routing-manager.
- Выполнение следующих команд к подам и сервисам в пространстве имён модулей из подсистемы `networking`:

- `kubectl attach`;
- `kubectl exec`;
- `kubectl port-forward`;
- `kubectl proxy`.

Пример назначения административных прав пользователю в рамках пространства имён



Пример использует экспериментальную ролевую модель.

Для назначения прав на управление ресурсами приложений в рамках пространства имён, но без возможности настройки модулей DKP, используйте роль `d8:use:role:admin` в `RoleBinding` в соответствующем пространстве имён.

Пример назначения прав разработчику приложений (User `app-developer`) в пространстве имён `myapp`:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:use:role:admin
  apiGroup: rbac.authorization.k8s.io

```

Список прав, которые получит пользователь

В рамках пространства имён `myapp` пользователю будут доступны следующие права:

- Просмотр, изменение, удаление и создание ресурсов Kubernetes. Например, следующих ресурсов (список не полный):
 - Certificate;
 - CertificateRequest;
 - ConfigMap;
 - ControllerRevision;
 - CronJob;
 - DaemonSet;
 - Deployment;
 - Event;
 - HorizontalPodAutoscaler;
 - Ingress;
 - Issuer;
 - Job;
 - Lease;
 - LimitRange;
 - NetworkPolicy;
 - PersistentVolumeClaim;
 - Pod;
 - PodDisruptionBudget;
 - ReplicaSet;
 - ReplicationController;
 - ResourceQuota;
 - Role;
 - RoleBinding;
 - Secret;
 - Service;
 - ServiceAccount;
 - StatefulSet;
 - VerticalPodAutoscaler;
 - VolumeSnapshot.
- Просмотр, изменение, удаление и создание следующих ресурсов модулей DKP:
 - DexAuthenticator;
 - DexClient;
 - PodLoginConfig.
- Выполнение следующих команд к подам и сервисам:
 - `kubectl attach`;
 - `kubectl exec`;
 - `kubectl port-forward`;
 - `kubectl proxy`.

Пример ClusterAuthorizationRule

```

apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
    - kind: User
      name: some@example.com
    - kind: ServiceAccount
      name: gitlab-runner-deploy
      namespace: d8-service-accounts
    - kind: Group
      name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # Опция доступна только при включенном режиме enableMultiTenancy.
  allowAccessToSystemNamespaces: false
  # Опция доступна только при включенном режиме enableMultiTenancy.
  namespaceSelector:
    labelSelector:
      matchExpressions:
        - key: stage
          operator: In
          values:
            - test
            - review
      matchLabels:
        team: frontend

```

Субъект типа `User` или `Group` сопоставляется по имени с идентичностью пользователя из токена. При этом локально управляемая идентичность не отличается от полученной от внешнего провайдера аутентификации. По этой причине нельзя создать ресурс `User` со значением `spec.email` или `Group` со значением `spec.name`, совпадающим с субъектом существующего правила `ClusterAuthorizationRule`. Это предотвращает незаметное предоставление прав идентичности.

Если совпадение сделано намеренно, например, если правило `ClusterAuthorizationRule` написано заранее, установите на `User` или `Group` аннотацию `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"`.

Для субъектов типа `User` указывайте email в нижнем регистре. Он записывается в токен в нижнем регистре, поэтому, например, субъект `Admin@Example.com` не совпадёт с `admin@example.com`. Имена субъектов типа `Group` сравниваются точно, поскольку имена групп не приводятся к нижнему регистру.

Ограничение действует только при создании ресурсов `User` и `Group` и не запрещает добавлять существующих пользователей и группы в `ClusterAuthorizationRule`. Если соответствующий `User` или `Group` уже существует, после добавления его в правило права будут предоставлены сразу.

Пример выдачи прав на все неймспейсы

i Пример относится к текущей ролевой модели.

В режиме `multi-tenancy` (`userAuthz.enableMultiTenancy`) ограничение доступа по неймспейсам задаётся полями ресурса `ClusterAuthorizationRule`.

Варианты настройки:

- **Разрешить доступ во все неймспейсы, включая системные** (`kube-*`, `d8-*`, `loghouse`, `default` и т. п.). Чтобы разрешить доступ во все неймспейсы, включая системные, укажите `spec.namespaceSelector.matchAny: true`. Если параметр `namespaceSelector` задан, значения `limitNamespaces` и `allowAccessToSystemNamespaces` **игнорируются**.

Пример:

```

apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: all-namespaces-including-system
spec:
  subjects:
    - kind: User
      name: user@example.com
  accessLevel: Editor
  namespaceSelector:
    matchAny: true

```

- **Разрешить доступ во все неймспейсы, кроме системных.** Чтобы разрешить пользователю доступ во все неймспейсы, кроме системных, при создании `ClusterAuthorizationRule` не указывайте ни `namespaceSelector`, ни `limitNamespaces`, ни `allowAccessToSystemNamespaces`. Перечень системных неймспейсов — в описании полей CR.



Уровень доступа `SuperAdmin` **не снимает** ограничения по неймспейсам, заданные в параметрах `namespaceSelector` и `limitNamespaces`. При необходимости предоставления доступа ко всем неймспейсам задайте область явно, в том числе через `namespaceSelector.matchAny`.

Если одному субъекту соответствует несколько ресурсов `ClusterAuthorizationRule`, набор разрешённых неймспейсов **объединяется**; эффективный `accessLevel` — **самый сильный** среди всех подходящих правил. Подробнее — в FAQ.



Ограничения по неймспейсам из `ClusterAuthorizationRule` реализованы в цепочке авторизации с вебхуком. Если вебхук недоступен, эти ограничения **не применяются**, пока вебхук снова не станет доступен. Подробнее — в описании модуля.

Экспериментальная ролевая модель

Роли `d8:use:role:*` назначаются только через `RoleBinding` в **конкретном** неймспейсе — отдельное `RoleBinding` нужно на каждый неймспейс (или автоматизируйте выдачу). Роли `d8:manage:*` не распространяются на неймспейсы пользовательских приложений — только на системные (`d8-*`, `kube-*`) в рамках подсистемы.

Создание пользователя

В Kubernetes есть две категории пользователей:

- `ServiceAccount`’ы, учёт которых ведёт сам Kubernetes через API.
- Остальные пользователи и группы, учёт которых ведёт не сам Kubernetes, а некоторый внешний софт, который настраивает администратор кластера, — существует множество механизмов аутентификации и, соответственно, множество способов заводить пользователей. В настоящий момент поддерживаются способы аутентификации:
 - Через модуль `user-authn`. Модуль поддерживает следующие внешние провайдеры и протоколы аутентификации: GitHub, GitLab, Atlassian Crowd, BitBucket Cloud, Crowd, LDAP, OIDC. Подробнее — в документации модуля `user-authn`.
 - С помощью сертификатов.

При выпуске сертификата для аутентификации нужно указать в нём имя (`CN=<имя>`), необходимое количество групп (`O=<группа>`) и подписать его с помощью корневого CA-кластера. Именно этим механизмом вы аутентифицируетесь в кластере, когда, например, используете `kubectl` на master-узле.

Создание `ServiceAccount` для сервера и предоставление ему доступа

Создание `ServiceAccount` с доступом к Kubernetes API может потребоваться, например, при настройке развёртывания приложений через CI-системы.

1. Создайте `ServiceAccount`, например в пространстве имён `d8-service-accounts`:

```
d8 k create -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: d8-service-accounts
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: d8-service-accounts
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

2. Дайте необходимые ServiceAccount-права (используя custom resource ClusterAuthorizationRule):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-deploy
spec:
  subjects:
    - kind: ServiceAccount
      name: gitlab-runner-deploy
      namespace: d8-service-accounts
  accessLevel: SuperAdmin
  # Опция доступна только при включенном режиме enableMultiTenancy.
  allowAccessToSystemNamespaces: true
EOF
```

Если в конфигурации Deckhouse включён режим мультитенантности (в параметре `enableMultiTenancy`), настройте доступные для ServiceAccount пространства имён (в параметре `namespaceSelector`).

3. Определите значения переменных (они будут использоваться далее), выполнив следующие команды (**подставьте свои значения**):

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy.my-cluster
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config
```

4. Сгенерируйте секцию `cluster` в файле конфигурации `kubectl`:

Используйте один из следующих вариантов доступа к API-серверу кластера:

- Если есть прямой доступ до API-сервера:

1. Получите сертификат CA-кластера Kubernetes:

```
d8 k get cm kube-root-ca.crt -o jsonpath='{ .data.ca\.crt }' > /tmp/ca.crt
```

2. Сгенерируйте секцию `cluster` (используется IP-адрес API-сервера для доступа):


```
d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
  --server=https://$(d8 k get ep kubernetes -o json | jq -rc '.subsets[0] | "\
  (.addresses[0].ip):\(.ports[0].port)') \
  --certificate-authority=/tmp/ca.crt \
  --kubeconfig=$FILE_NAME
```

- Если прямого доступа до API-сервера нет, используйте один следующих вариантов:
 - включите доступ к API-серверу через Ingress-контроллер (параметр `publishAPI` и укажите адреса, с которых будут идти запросы (параметр `whitelistSourceRanges`);
 - укажите адреса, с которых будут идти запросы, в отдельном Ingress-контроллере (параметр `acceptRequestsFrom`).

- Если используется непубличный CA:

1. Получите сертификат CA из секрета с сертификатом, который используется для домена `api.%s`:

```
d8 k -n d8-user-authn get secrets -o json \
  $(d8 k -n d8-user-authn get ing kubernetes-api -o jsonpath="
  {.spec.tls[0].secretName}") \
  | jq -rc '.data."ca.crt" // .data."tls.crt"' \
  | base64 -d > /tmp/ca.crt
```

2. Сгенерируйте секцию `cluster` (используется внешний домен и CA для доступа):

```
d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
  --server=https://$(d8 k -n d8-user-authn get ing kubernetes-api -o json | jq
  '.spec.rules[].host' -r) \
  --certificate-authority=/tmp/ca.crt \
  --kubeconfig=$FILE_NAME
```

- Если используется публичный CA. Сгенерируйте секцию `cluster` (используется внешний домен для доступа):

```
d8 k config set-cluster $CLUSTER_NAME \
  --server=https://$(d8 k -n d8-user-authn get ing kubernetes-api -o json | jq
  '.spec.rules[].host' -r) \
  --kubeconfig=$FILE_NAME
```

5. Сгенерируйте секцию `user` с токеном из секрета `ServiceAccount` в файле конфигурации `kubectl`:

```
d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n d8-service-accounts get secret gitlab-runner-deploy-token -o json | jq -r
  '.data["token"]' | base64 -d) \
  --kubeconfig=$FILE_NAME
```

6. Сгенерируйте контекст в файле конфигурации `kubectl`:

```
d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME
```

7. Установите сгенерированный контекст как используемый по умолчанию в файле конфигурации `kubectl`:

```
d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

Создание пользователя с помощью клиентского сертификата



Этот способ рекомендуется использовать для системных нужд (аутентификация kubelet'ов, компонентов control plane и пр.). Если нужно создать «обычного» пользователя (например, с доступом через консоль, `kubectl` и т.д.), используйте генерацию `kubeconfig`.

При создании пользователя с помощью клиентского сертификата можно использовать OpenSSL или Kubernetes API (объект `CertificateSigningRequest`).



Сертификаты, выпущенные любым из этих способов, отозвать нельзя. В случае компрометации сертификата потребуется убрать все права этого пользователя (это может быть сложно, если пользователь добавлен в какие-нибудь группы: придётся также удалять все соответствующие группы).

Создание пользователя с помощью сертификата, выпущенного через OpenSSL



При использовании этого способа учитывайте риски безопасности.

`ca.crt` и `ca.key` не должны покидать master-узел: подписывайте CSR только на нём.

При подписании CSR вне master-узла есть риск компрометации корневого сертификата кластера.

Особенности этого способа:

- Клиентский сертификат должен подписываться на master-узле, чтобы не допустить компрометации кластерного сертификата.
- Необходим доступ к СА-ключу кластера (`ca.key`). Подписывать сертификаты может только администратор кластера.

Чтобы создать пользователя с помощью клиентского сертификата, выпущенного через OpenSSL, выполните следующие шаги:

1. Получите корневой сертификат кластера (`ca.crt` и `ca.key`).
2. Сгенерируйте ключ пользователя:

```
openssl ecparam -name prime256v1 -genkey -out myuser.key
```

3. Создайте CSR, указав в нём имя пользователя `myuser`, который состоит в группах `mygroup1` и `mygroup2`:

```
openssl req -new -key myuser.key -out myuser.csr -subj
"/CN=myuser/O=mygroup1/O=mygroup2"
```

4. Загрузите созданный на предыдущем шаге CSR (в этом примере — `myuser.csr`) на master-узел и подпишите его корневым сертификатом кластера. Пример команды для подписания CSR на мастер-узле (убедитесь, что в команде указаны верные для вашего случая пути к `myuser.csr`, `ca.crt` и `ca.key`):

```
openssl x509 -req -in myuser.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out myuser.crt -days 10
```

Полученный сертификат можно указывать в конфигурационном файле:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
    server: https://<хост кластера>:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: myuser
    name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Создание пользователя с помощью сертификата, выпущенного через Kubernetes API

Это более безопасный способ, т.к. для подписания сертификата используется специальный API `kubernetes`.

Особенности этого способа:

- Подписание сертификата через Kubernetes API: CSR отправляется на подпись через API и прямой доступ к `ca.key` не требуется.
- Выпускать клиентские сертификаты может не только администратор кластера. Право на создание CSR и их подписание можно назначить определенному пользователю.

Чтобы создать пользователя с помощью клиентского сертификата, выпущенного через Kubernetes API, выполните следующие шаги:

1. Сгенерируйте ключ пользователя:

```
openssl ecparam -name prime256v1 -genkey -out myuser.key
```

2. Создайте CSR, указав в нём имя пользователя `myuser`, который состоит в группах `mygroup1` и `mygroup2`:

```
openssl req -new -key myuser.key -out myuser.csr -subj
"/CN=myuser/O=mygroup1/O=mygroup2"
```

3. Создайте манифест объекта `CertificateSigningRequest` и сохраните его в файл (в этом примере — `csr.yaml`):

i В поле `request` укажите содержимое CSR, созданного на предыдущем этапе, закодированное в Base64.

```

apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: demo-client-cert
spec:
  request: # CSR в Base64
  signerName: "kubernetes.io/kube-apiserver-client"
  expirationSeconds: 7200
  usages:
    - "digital signature"
    - "client auth"

```

4. Примените манифест, чтобы создать запрос на подпись сертификата:

```
d8 k apply -f csr.yaml
```

5. Убедитесь, что сертификат подтвержден:

```
d8 k get csr demo-client-cert
```

Если сертификат подтвержден, в колонке `CONDITION` у него будет значение `Approved, Issued`. Пример вывода:

NAME	AGE	SIGNERNAME	REQUESTOR
demo-client-cert	8m24s	kubernetes.io/kube-apiserver-client	kubernetes-admin
Approved, Issued			120m

Если сертификат не подтвердился автоматически, подтвердите его:

```
d8 k certificate approve demo-client-cert
```

После этого убедитесь, что сертификат подтвержден.

6. Извлеките закодированный сертификат из CSR с именем `demo-client-cert`, декодируйте его из Base64 и сохраните в файл (в этом примере — `myuser.crt`), созданный на шаге 2:

```
d8 k get csr demo-client-cert -ojsonpath="{.status.certificate}" | base64 -d > myuser.crt
```

Полученный сертификат необходимо указать в конфигурационном файле:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
  server: https://<хост кластера>:6443
  name: kubernetes
contexts:
- context:
  cluster: kubernetes
  user: myuser
  name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Предоставление доступа созданному пользователю

Для предоставления доступа созданному пользователю создайте `ClusterAuthorizationRule`.

Пример `ClusterAuthorizationRule`:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: myuser
spec:
  subjects:
  - kind: User
    name: myuser
  accessLevel: PrivilegedUser
  portForwarding: true
```

Настройка `kube-apiserver` для работы в режиме multi-tenancy

Режим multi-tenancy, позволяющий ограничивать доступ к пространству имён, включается параметром `enableMultiTenancy` модуля.

Работа в режиме multi-tenancy требует включения плагина авторизации Webhook и выполнения настройки `kube-apiserver`. Все необходимые для работы режима multi-tenancy действия **выполняются автоматически** модулем `control-plane-manager`, никаких ручных действий не требуется.

Изменения манифеста `kube-apiserver`, которые произойдут после включения режима multi-tenancy:

- настройка цепочки авторизации через structured authorization configuration (-- authorization-config=/etc/kubernetes/deckhouse/extra-files/authorization-config.yaml) с авторизаторами Node, Webhook и RBAC;
- создание kubeconfig для authorization webhook по пути /etc/kubernetes/deckhouse/extra-files/webhook-config.yaml;
- добавление volumeMounts:

```
- name: authorization-webhook-config
  mountPath: /etc/kubernetes/authorization-webhook-config.yaml
  readOnly: true
```

- добавление `volumes`:

```
- name: authorization-webhook-config
  hostPath:
    path: /etc/kubernetes/authorization-webhook-config.yaml
    type: FileOrCreate
```

Как проверить, что у пользователя есть доступ?

Необходимо выполнить следующую команду, в которой будут указаны:

- `resourceAttributes` (как в RBAC) — к чему мы проверяем доступ;
- `user` — имя пользователя;
- `groups` — группы пользователя.

i При совместном использовании с модулем `user-authn` группы и имя пользователя можно посмотреть в логах Dex — `d8 k -n d8-user-authn logs -l app=dex` (видны только при авторизации).

```
cat <<EOF | 2>&1 d8 k create --raw
/apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

В результате увидим, есть ли доступ и на основании какой роли:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by ClusterRoleBinding \"system:kube-controller-manager\"
of ClusterRole \"system:kube-controller-manager\" to User \"system:kube-controller-
manager\""
```

Если в кластере включён режим **multi-tenancy**, нужно выполнить ещё одну проверку, чтобы убедиться, что у пользователя есть доступ в пространство имён:

```
cat <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-
config.yaml create --raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

```
{
  "allowed": false
}
```

Сообщение `allowed: false` значит, что вебхук не блокирует запрос. В случае блокировки запроса вебхуком вы получите, например, следующее сообщение:

```
{
  "allowed": false,
  "denied": true,
  "reason": "making cluster scoped requests for namespaced resources are not
allowed"
}
```

Настройка прав высокоуровневых ролей

Если требуется добавить прав для определённой высокоуровневой роли, достаточно создать `ClusterRole` с аннотацией `user-authz.deckhouse.io/access-level: <AccessLevel>`.

Модуль автоматически выставляет на такие `ClusterRole` лейбл `user-authz.deckhouse.io/access-level` и включает их правила в агрегированную `ClusterRole` `user-authz:<access-level>:custom`. Лейбл служебный: модуль приводит его в соответствие с аннотацией и снимает, если аннотации нет, поэтому задавать его вручную не нужно. В журнале аудита доступ, выданный через аннотированную `ClusterRole`, приписывается этой агрегированной роли.

Пример:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete

```

FAQ

Как создать пользователя?

Создание пользователя.

Как ограничить права пользователю конкретными пространствами имён?

Чтобы ограничить права пользователя конкретными пространствами имён в экспериментальной ролевой модели, используйте в `RoleBinding` use-роль с соответствующим уровнем доступа. Пример....

В текущей ролевой модели используйте параметры `namespaceSelector` или `limitNamespaces` (устарел) в кастомном ресурсе `ClusterAuthorizationRule`.

Что, если два `ClusterAuthorizationRules` подходят для одного пользователя?

В примере пользователь `jane.doe@example.com` состоит в группе `administrators`. Созданы два `ClusterAuthorizationRules`:


```

apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: jane
spec:
  subjects:
    - kind: User
      name: jane.doe@example.com
  accessLevel: User
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
    - kind: Group
      name: administrators
  accessLevel: ClusterAdmin
  namespaceSelector:
    labelSelector:
      matchExpressions:
        - key: env
          operator: In
          values:
            - prod
            - stage

```

1. `jane.doe@example.com` имеет право запрашивать и просматривать объекты среди всех пространств имён, помеченных `env=review`.
2. `Administrators` могут запрашивать, редактировать, получать и удалять объекты на уровне кластера и из пространств имён, помеченных `env=prod` и `env=stage`.

Так как для `Jane Doe` подходят два правила, необходимо провести вычисления:

- `Jane Doe` будет иметь самый сильный `accessLevel` среди всех подходящих правил — `ClusterAdmin`.
- Опции `namespaceSelector` будут объединены так, что `Jane Doe` будет иметь доступ в пространства имён, помеченные лейблом `env` со значением `review`, `stage` или `prod`.



Если есть правило без опции `namespaceSelector` и без опции `limitNamespaces` (устаревшая), это значит, что доступ разрешён во все пространства имён, кроме системных, что повлияет на результат вычисления доступных пространств имён для пользователя.

Как расширить роли или создать новую?

Экспериментальная ролевая модель построена на принципе агрегации, она собирает более мелкие роли в более обширные, тем самым предоставляя лёгкие способы расширения модели собственными ролями.

Создание новой роли подсистемы

Предположим, что текущие подсистемы не подходят под ролевое распределение в компании и требуется создать новую подсистему, которая будет включать в себя роли из подсистемы `deckhouse`, подсистемы `kubernetes` и модуля `user-authn`.

Для решения этой задачи создайте следующую роль:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
rules: []

```

В начале указаны лейблы для новой роли:

- показывает, какую роль хук должен использовать при создании use ролей:

```
rbac.deckhouse.io/use-role: admin
```

- показывает, что роль должна обрабатываться как manage-роль:

```
rbac.deckhouse.io/kind: manage
```

i Этот лейбл обязателен.

- показывает, что роль является ролью подсистемы, и обрабатываться будет соответственно:

```
rbac.deckhouse.io/level: subsystem
```

- указывает подсистему, за которую отвечает роль:

```
rbac.deckhouse.io/subsystem: custom
```

- позволяет `manage:all`-роли агрегировать эту роль в себя:

```
rbac.deckhouse.io/aggregate-to-all-as: manager
```

Далее указаны селекторы, именно они реализуют агрегацию:

- агрегирует роль менеджера из подсистемы `deckhouse`:

```

rbac.deckhouse.io/kind: manage
rbac.deckhouse.io/aggregate-to-deckhouse-as: manager

```

- агрегирует все правила от модуля user-authn:

```

rbac.deckhouse.io/kind: manage
module: user-authn

```

Таким образом роль получает права от подсистем `deckhouse`, `kubernetes` и от модуля `user-authn`.

Особенности:

- ограничений на имя роли нет, но для читаемости лучше использовать этот стиль;
- `use`-роли будут созданы в пространстве имён агрегированных подсистем и модуля, тип роли выбран лейблом.

Расширение пользовательской роли

Например, в кластере появился новый кластерный (пример для `manage`-роли) CRD-объект — `MySuperResource`, и нужно дополнить собственную роль из примера выше правами на взаимодействие с этим ресурсом.

Первым делом нужно дополнить роль новым селектором:

```
rbac.deckhouse.io/kind: manage
rbac.deckhouse.io/aggregate-to-custom-as: manager
```

Этот селектор позволит агрегировать роли к новой подсистеме через указание этого лейбла. После добавления нового селектора роль будет выглядеть так:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-custom-as: manager
rules: []
```

Далее нужно создать новую роль, в которой следует определить права для нового ресурса. Например, только чтение:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-custom-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Роль дополнит своими правами роль подсистемы, дав права на просмотр нового объекта.

Особенности:

- ограничений на имя роли нет, но для читаемости лучше использовать этот стиль.

Расширение существующих `manage subsystem`-ролей

Если необходимо расширить существующую роль, нужно выполнить те же шаги, что и в пункте выше, но изменив лейблы и название роли.

Пример для расширения роли менеджера из подсистемы `deckhouse (d8:manage:deckhouse:manager)`:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustommodule:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Таким образом новая роль расширит роль `d8:manage:deckhouse`.

Расширение `manage subsystem`-ролей с добавлением нового пространства имён

Если необходимо добавить новое пространство имён (для создания в нём use-роли с помощью хука), потребуется добавить лишь один лейбл:

```
"rbac.deckhouse.io/namespace": namespace
```

Этот лейбл сообщает хуку, что в этом пространстве имён нужно создать use-роль:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/namespace: namespace
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Хук мониторит `ClusterRoleBinding` и при создании биндинга ходит по всем `manage`-ролям, чтобы найти все объединенные в них роли с помощью проверки правила агрегации. Затем он берёт пространство имён из лейбла `rbac.deckhouse.io/namespace` и создает `use`-роль в этом пространстве имён.

Расширение существующих `use`-ролей

Если ресурс принадлежит пространству имён, необходимо расширить `use`-роль вместо `manage`-роли. Разница лишь в лейблах и имени:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-kubernetes-as: user
    rbac.deckhouse.io/kind: use
  name: custom:use:capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch

```

Эта роль дополнит роль `d8:use:role:user:kubernetes`.

Модуль `vertical-pod-autoscaler`

Vertical Pod Autoscaler (VPA) — это инфраструктурный сервис, который позволяет не выставлять точные `resource requests`, если неизвестно, сколько ресурсов необходимо контейнеру для работы. При использовании VPA и включении соответствующего режима работы `resource requests` выставляются автоматически на основе потребления ресурсов (полученных данных из Prometheus). Как вариант, возможно только получать рекомендации по ресурсам, без их автоматического изменения.

У VPA есть следующие режимы работы:

- **"Auto"** (default) — в данный момент режимы работы `Auto` и `Recreate` делают одно и то же. Однако, когда в Kubernetes появится `Pod in-place resource update`, этот режим будет делать именно его.
- **"Recreate"** — режим разрешает VPA изменять ресурсы у запущенных подов (перезапускать их при работе). В случае работы одного пода (`replicas: 1`) это приведет к недоступности сервиса на время рестарта. В данном режиме VPA не пересоздает поды,

которые были созданы без контроллера.

- "Initial" — VPA изменяет ресурсы подов только при создании подов, но не во время работы.
- "Off" — VPA не изменяет автоматически никакие ресурсы. В данном случае, если есть VPA с таким режимом работы, мы можем посмотреть, какие ресурсы рекомендует поставить VPA (`d8 k describe vpa <vpa-name>`).



Режим Auto устарел. Его поддержка будет прекращена в будущих версиях DKP. Перейдите на один из поддерживаемых режимов работы.

Ограничения VPA:

- Обновление ресурсов запущенных подов — это экспериментальная возможность VPA. Каждый раз, когда VPA обновляет `resource requests` пода, под пересоздается. Под может быть создан на другом узле.
- VPA **не должен использоваться с HPA по CPU и памяти** в данный момент. Однако VPA можно использовать с HPA на `custom/external metrics`.
- VPA реагирует почти на все `out-of-memory events`, но не гарантирует реакцию.
- Производительность VPA не тестировалась на огромных кластерах.
- Рекомендации VPA могут превышать доступные ресурсы в кластере, что **может приводить к подам в состоянии Pending**.
- Использование нескольких VPA-ресурсов над одним подом может привести к неопределенному поведению.
- В случае удаления VPA или его «выключения» (режим `Off`) изменения, внесенные ранее VPA, не сбрасываются, а остаются в последнем измененном значении. Из-за этого может возникнуть путаница, что в Helm будут описаны одни ресурсы, при этом в контроллере тоже будут описаны одни ресурсы, но реально у подов ресурсы будут совсем другие и может сложиться впечатление, что они взялись «непонятно откуда».



При использовании VPA настоятельно рекомендуется использовать Pod Disruption Budget.

Grafana dashboard

Представлено на дашбордах:

- `Main / Namespace`, `Main / Namespace / Controller`, `Main / Namespace / Controller / Pod` — столбец VPA `type` показывает значение `updatePolicy.updateMode`;
- `Main / Namespaces` — столбец VPA `%` показывает процент подов с включенным VPA.

Архитектура Vertical Pod Autoscaler

VPA состоит из 3 компонентов:

- `Recommender` — мониторит настоящее (делая запросы в Metrics API, который реализован в модуле `prometheus-metrics-adapter`) и прошлое потребление ресурсов (делая запросы в Trickster перед Prometheus) и предоставляет рекомендации по CPU и памяти для контейнеров.
- `Updater` — проверяет, что у подов с VPA выставлены корректные ресурсы, если нет — убивает эти поды, чтобы контроллер пересоздал поды с новыми `resource requests`.
- `Admission Plugin` — задает `resource requests` при создании новых подов (контроллером или из-за активности Updater'a).

При изменении ресурсов компонентом Updater это происходит с помощью Eviction API, поэтому учитываются Pod Disruption Budget для обновляемых подов.



Настройки

VPA работает не с контроллером пода, а с самим подом, измеряя и изменяя параметры его контейнеров. Вся настройка происходит с помощью `custom resource's VerticalPodAutoscaler`.

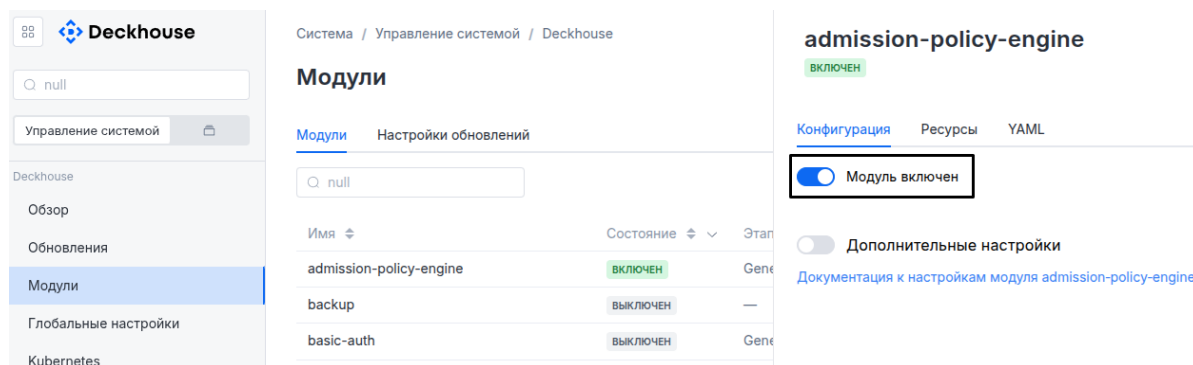
В общем случае конфигурация модуля не требуется. У модуля есть только настройки `nodeSelector/tolerations`.

Модуль **включен** по умолчанию в наборах модулей: `Default`, `Managed`. Модуль **выключен** по умолчанию в наборе модулей `Minimal`.

Явно включить или выключить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.** В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `vertical-pod-autoscaler`, включите (или выключите) переключатель «Модуль включен». Сохраните изменения.

Пример:



- **С помощью Deckhouse CLI (d8).**

Используйте команду `d8 system module enable` для включения модуля, или `d8 system module disable` для выключения модуля (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Пример включения модуля `vertical-pod-autoscaler`:

```
d8 system module enable vertical-pod-autoscaler
```

- **С помощью ModuleConfig `vertical-pod-autoscaler`.**

Установите `spec.enabled` в `true` или `false` в ModuleConfig `vertical-pod-autoscaler` (создайте его, при необходимости).

Пример манифеста для включения модуля `vertical-pod-autoscaler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  enabled: true
```

Как настроить модуль...

Настроить модуль можно одним из следующих способов:

- **С помощью веб-интерфейса Deckhouse.**

В разделе «Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `vertical-pod-autoscaler`, включите переключатель «Дополнительные настройки». Заполните необходимые поля формы на вкладке «Конфигурация», или укажите настройки модуля в формате YAML на вкладке «YAML», **не включая секцию `settings`**. Сохраните изменения.

Пример:

Конфигурация Ресурсы YAML

☒ Модуль включен

☒ Дополнительные настройки

[Документация к настройкам модуля admission-policy-engine](#)

Версия настроек

1

Конфигурация YAML

Вы также можете отредактировать объект ModuleConfig `vertical-pod-autoscaler` на вкладке «YAML» в окне настроек модуля («Система» → «Управление системой» → «Deckhouse» → «Модули», откройте модуль `vertical-pod-autoscaler`), указав версию схемы в параметре `spec.version` и необходимые параметры модуля в секции `spec.settings`.

- **С помощью Deckhouse CLI (d8)** (требуется Deckhouse CLI (d8), настроенный на работу с кластером).

Отредактируйте существующий ModuleConfig `vertical-pod-autoscaler` (подробнее о настройке Deckhouse читайте в документации), выполнив следующую команду:

```
d8 k edit mc vertical-pod-autoscaler
```

Внесите необходимые изменения в секцию `spec.settings`. При необходимости укажите версию схемы в параметре `spec.version`. Сохраните изменения.

Вы также можете создать файл манифеста ModuleConfig `vertical-pod-autoscaler`, используя пример ниже. Заполните секцию `spec.settings` необходимыми параметрами модуля. При необходимости укажите версию схемы в параметре `spec.version`.

Примените манифест с помощью следующей команды (укажите имя файла манифеста):

```
d8 k apply -f <FILENAME>
```

Пример файла манифеста ModuleConfig `vertical-pod-autoscaler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings: # Параметры модуля из раздела "Параметры" ниже.
```

Требования

К версиям других модулей:

- `prometheus`: любая версия.
- `prometheus-metrics-adapter`: любая версия.

Параметры

Версия схемы: 1

- **settings**

объект

- **settings.nodeSelector**

объект

Структура, аналогичная `spec.nodeSelector` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations**

массив объектов

Структура, аналогичная `spec.tolerations` пода Kubernetes.

Если значение не указано или указано `false`, будет использоваться автоматика.

- **settings.tolerations.effect**

строка

- **settings.tolerations.key**

строка

- **settings.tolerations.operator**

строка

- **settings.tolerations.tolerationSeconds**

целочисленный

- **settings.tolerations.value**

строка

Custom resources

i

Deckhouse Kubernetes Platform устанавливает CRD, но не удаляет их при отключении модуля. Если вам больше не нужны созданные CRD, удалите их.

VerticalPodAutoscaler

Short names: vpa

Scope: Namespaced

- v1
- v1beta2

Описывает конфигурацию работы механизма Vertical Pod Autoscaler для указанного контроллера.

Vertical Pod Autoscaler автоматически управляет ресурсами подов контроллера на основании исторических и текущих данных по утилизации ресурсов.

- **apiVersion**

строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации.

- **kind**

строка

Значение, представляющее REST-ресурс, который описывает данный объект. Серверы могут определить его из конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации.

- **metadata**

объект

- **spec**

объект

Обязательный параметр

Описание поведения автоскейлера.

Больше информации в документации.

- **spec.recommenders**

массив объектов

Механизм рекомендаций, отвечающий за создание рекомендаций для данного объекта. Список должен быть пустым (тогда рекомендации создаст механизм по умолчанию) или содержать ровно один элемент.

`VerticalPodAutoscalerRecommenderSelector` указывает на конкретный механизм рекомендаций для Vertical Pod Autoscaler. В будущем он может передавать параметры этому механизму.

- **spec.recommenders.name**

строка

Обязательный параметр

Имя компонента, отвечающего за создание рекомендаций для данного объекта.

- **spec.resourcePolicy**

объект

Управляет тем, как автоскейлер вычисляет рекомендуемые ресурсы. Политика ресурсов может использоваться для установки ограничений на рекомендации для отдельных контейнеров.

Если какие-либо контейнеры необходимо исключить из получения рекомендаций VPA, это следует явно указать, установив режим в значение `Off` в параметре `containerPolicies`.

Если параметр не задан, автоскейлер будет вычислять рекомендуемые ресурсы для всех контейнеров в поде без дополнительных ограничений.

- **spec.resourcePolicy.containerPolicies**

массив объектов

Политики ресурсов для каждого контейнера.

Политика ресурсов контейнера контролирует, как автоскейлер вычисляет рекомендуемые ресурсы для конкретного контейнера.

- **spec.resourcePolicy.containerPolicies.containerName**

строка

Имя контейнера или `DefaultContainerResourcePolicy`, в этом случае политика применяется к контейнерам, для которых не указана собственная политика.

- **spec.resourcePolicy.containerPolicies.controlledResources**

массив строк

Указывает тип рекомендаций, которые будут вычисляться (и, возможно, применяться) VPA. Если не указано, используется значение по умолчанию: `ResourceCPU`, `ResourceMemory`.

По умолчанию: `["cpu", "memory"]`

- **Элемент массива**

строка

Имя, которое идентифицирует различные ресурсы в списке ресурсов (`ResourceList`).

- **spec.resourcePolicy.containerPolicies.controlledValues**

строка

Указывает, какие значения ресурсов должны контролироваться.

По умолчанию: `RequestsAndLimits`

Допустимые значения: `RequestsAndLimits`, `RequestsOnly`

- **spec.resourcePolicy.containerPolicies.maxAllowed**

объект

Указывает максимальное количество ресурсов, которое будет рекомендовано для контейнера. По умолчанию максимальное значение не установлено.

- **spec.resourcePolicy.containerPolicies.minAllowed**

объект

Указывает минимальное количество ресурсов, которое будет рекомендовано для контейнера. По умолчанию минимальное значение не установлено.

- **spec.resourcePolicy.containerPolicies.mode**

строка

Указывает, включен ли автоскейлер для контейнера.

По умолчанию: `Auto`

Допустимые значения: `Auto`, `Off`

- **spec.scope**

строка

Включает scoped-режим рекомендаций для `targetRef.kind=DaemonSet`. Значение должно быть ключом лейбла узла (например, `node.kubernetes.io/instance-type`).

В scoped-режиме рекомендации группируются по `status.groups[].scopeValue`. Для остальной нагрузки и для `DaemonSet` без `scope` используется обычное поле `status.recommendation`.

- **spec.targetRef**

объект

Обязательный параметр

Указывает на контроллер, управляющий набором подов, которые должен контролировать автоскейлер, например, `Deployment` или `StatefulSet`.

`VerticalPodAutoscaler` может быть нацелен на контроллер, который реализует подресурс масштабирования (набор подов извлекается из `ScaleStatus` контроллера) или на другие известные контроллеры (например, для `DaemonSet` набор подов читается из спецификации контроллера).

Если `VerticalPodAutoscaler` не может использовать указанный объект, он сообщит о состоянии `ConfigUnsupported`. Обратите внимание, что `VerticalPodAutoscaler` не требует полной реализации подресурса масштабирования — он не будет использовать его для изменения количества реплик. Единственное, что используется — это селектор лейблов, соответствующих подам, сгруппированным по целевому ресурсу.

- **spec.targetRef.apiVersion**

строка

Версия API связанного объекта.

- **spec.targetRef.kind**

строка

Обязательный параметр

Тип связанного объекта.

Больше информации в документации.

- **spec.targetRef.name**

строка

Обязательный параметр

Имя связанного объекта.

Больше информации в документации.

- **spec.updatePolicy**

объект

Описывает правила применения изменений к подам. Если не указано, все поля в `PodUpdatePolicy` устанавливаются в значения по умолчанию.

- **spec.updatePolicy.evictionRequirements**

массив объектов

Список требований для завершения работы пода (`EvictionRequirements`), которые должны быть выполнены, чтобы под завершил работу (`Eviction`). Если

указано несколько требований, все они должны быть выполнены для разрешения завершения работы.

Определяет одно условие, которое должно быть истинным для того, чтобы под завершил работу.

- **spec.updatePolicy.evictionRequirements.changeRequirement**
строка

Обязательный параметр

Описывает зависимость между новым целевым рекомендуемым значением для пода и его текущими запросами, а также тип изменения, необходимого для того, чтобы под завершил работу.

Допустимые значения: `TargetHigherThanRequests`,
`TargetLowerThanRequests`

- **spec.updatePolicy.evictionRequirements.resources**
массив строк

Обязательный параметр

Это список одного или нескольких ресурсов, к которым применяется условие. Если указано несколько ресурсов, условие для завершения работы считается выполненным, если хотя бы один из ресурсов удовлетворяет `changeRequirement`.

- **Элемент массива**
строка

Это имя, идентифицирующее различные ресурсы в списке ресурсов (`ResourceList`).

- **spec.updatePolicy.minReplicas**
целочисленный

Минимальное количество реплик, которые должны быть в живом состоянии, чтобы обновляющий процесс (Updater) попытался удалить под (с учётом других проверок, таких как PDB). Разрешены только положительные значения. Перезаписывает глобальный флаг `--min-replicas`.

- **spec.updatePolicy.updateMode**
строка

Режим работы контроллера VPA.

- `Auto` — режимы `Auto` и `Recreate` выполняют одинаковую функцию. Этот режим зарезервирован для обновления ресурсов пода в Kubernetes.
- `Recreate` — позволяет VPA изменять требования к ресурсам работающих подов (т.е. перезапускать их в процессе работы). Этот режим может привести к временной недоступности сервиса (из-за перезапуска), если у вас только одна реплика (`replicas: 1`). В этом режиме VPA не пересоздаёт поды, которые не были созданы контроллером.
- `Initial` — VPA изменяет ресурсы подов только при их запуске (но не во время работы подов).
- `Off` — VPA не предпринимает никаких действий для обновления запросов ресурсов для работающих контейнеров. Автоскейлер рассчитывает рекомендации и сохраняет их в поле статуса объекта VPA. Вы можете просмотреть рекомендации VPA с помощью следующей команды: `d8 k describe vpa <vpa-name>`.
- `InPlaceOrRecreate (GA)` — работает в версиях Kubernetes 1.33 и старше. VPA попытается изменить ресурсы пода без перезагрузки и при невозможности пересоздаст его.

По умолчанию: `InPlaceOrRecreate`

Допустимые значения: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **status**
объект

Текущая информация об автоскейлере.

- **status.conditions**
массив объектов

Набор условий, необходимых для того, чтобы автоскейлер мог масштабировать свою цель, и указывает, выполнены ли эти условия.

Описывает состояние VerticalPodAutoscaler в определённый момент времени.

- **status.conditions.lastTransitionTime**

строка

Время последнего изменения состояния условия с одного статуса на другой.

- **status.conditions.message**

строка

Сообщение, которое содержит объяснение и детали о переходе.

- **status.conditions.reason**

строка

Причина последнего перехода состояния.

- **status.conditions.status**

строка

Обязательный параметр

Состояние условия (`True`, `False`, `Unknown`).

- **status.conditions.type**

строка

Обязательный параметр

Описывает текущее состояние условия.

- **status.groups**

массив объектов

Сгруппированные рекомендации по значениям `spec.scope` для scoped DaemonSet.

В этом режиме `status.groups` является источником истины, а `status.recommendation` не заполняется.

- **status.groups.containerRecommendations**

массив объектов

Рекомендации по контейнерам для группы.

В scoped-режиме поле хранится в компактном виде (в первую очередь `containerName` и `target`).

- **status.groups.containerRecommendations.containerName**

строка

Имя контейнера.

- **status.groups.containerRecommendations.lowerBound**

объект

Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

- **status.groups.containerRecommendations.scope**

строка

Необязательное поле совместимости.

Для scoped DaemonSet фактическая область рекомендации определяется через `spec.scope` + `scopeValue` группы.

- **status.groups.containerRecommendations.target**

объект

Обязательный параметр

Recommended amount of resources. Observes `ContainerResourcePolicy`.

- **status.groups.containerRecommendations.uncappedTarget**

объект

The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`.

May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than `MinAllowed` or higher than `MaxAllowed`). Used only as status indication, will not affect actual resource assignment.

- **status.groups.containerRecommendations.upperBound**

объект

Maximum recommended amount of resources.

Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

- **status.groups.scopeValue**

строка

Обязательный параметр

Значение лейбла `spec.scope` узла, для которого рассчитана группа.

Специальное значение `__absent__` означает, что у узла отсутствует ключ лейбла `spec.scope`.

- **status.recommendation**

объект

Последнее вычисленное количество ресурсов, рекомендованных автоскейлером для контролируемых подов.

Используется для обычного режима (Deployment/StatefulSet/DaemonSet без `spec.scope`).

- **status.recommendation.containerRecommendations**

массив объектов

Ресурсы, рекомендованные автоскейлером для каждого контейнера.

Рекомендации по ресурсам, вычисленные автоскейлером для конкретного контейнера. Учитывает политику ресурсов контейнера, если она указана в спецификации. В частности, рекомендация не генерируется для контейнеров, у которых установлен режим `ContainerScalingMode` равный 'Off'.

- **status.recommendation.containerRecommendations.containerName**

строка

Имя контейнера.

- **status.recommendation.containerRecommendations.lowerBound**

объект

Минимальное рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`. Это количество ресурсов не гарантирует, что приложение будет работать стабильно, однако использование меньшего количества ресурсов, скорее всего, окажет значительное влияние на производительность и доступность.

- **status.recommendation.containerRecommendations.scope**

строка

Связка `<ключ scope>=<значение>`.

- **status.recommendation.containerRecommendations.target**

объект

Обязательный параметр

Рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`.

- **status.recommendation.containerRecommendations.uncappedTarget**

объект

Наиболее ранняя рекомендованная цель ресурсов, вычисленная автоскейлером для контролируемых подов, основанная только на фактическом использовании ресурсов, без учета `ContainerResourcePolicy`. Может отличаться от рекомендации, если фактическое использование ресурсов приводит к нарушению `ContainerResourcePolicy` (меньше чем `MinAllowed` или больше чем `MaxAllowed`). Используется только как индикатор состояния, не влияет на фактическое назначение ресурсов.

- **status.recommendation.containerRecommendations.upperBound**
объект

Максимально рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`. Ресурсы, выделенные сверх этого значения, скорее всего, будут потрачены впустую. Это значение может быть больше, чем максимальное количество ресурсов, которое приложение фактически способно потреблять.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Описывает конфигурацию работы механизма Vertical Pod Autoscaler для указанного контроллера.

Vertical Pod Autoscaler автоматически управляет ресурсами подов контроллера на основании исторических и текущих данных по утилизации ресурсов.

- **apiVersion**
строка

Определяет версионированную схему этого представления объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации.

- **kind**
строка

Значение, представляющее REST-ресурс, который описывает данный объект. Серверы могут определить его из конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации.

- **metadata**
объект
- **spec**
объект

Обязательный параметр

Описание поведения автоскейлера.

Больше информации в документации.

- **spec.recommenders**
массив объектов

Механизм рекомендаций, отвечающий за создание рекомендаций для данного объекта. Список должен быть пустым (тогда рекомендации создаст механизм по умолчанию) или содержать ровно один элемент.

`VerticalPodAutoscalerRecommenderSelector` указывает на конкретный механизм рекомендаций для Vertical Pod Autoscaler. В будущем он может передавать параметры этому механизму.

- **spec.recommenders.name**
строка

Обязательный параметр

Имя компонента, отвечающего за создание рекомендаций для данного объекта.

- **spec.resourcePolicy**
объект

Управляет тем, как автоскейлер вычисляет рекомендуемые ресурсы. Политика ресурсов может использоваться для установки ограничений на рекомендации для отдельных контейнеров.

Если какие-либо контейнеры необходимо исключить из получения рекомендаций VPA, это следует явно указать, установив режим в значение `Off` в параметре `containerPolicies`.

Если параметр не задан, автоскейлер будет вычислять рекомендуемые ресурсы для всех контейнеров в поде без дополнительных ограничений.

- **spec.resourcePolicy.containerPolicies**

массив объектов

Политики ресурсов для каждого контейнера.

Политика ресурсов контейнера контролирует, как автоскейлер вычисляет рекомендуемые ресурсы для конкретного контейнера.

- **spec.resourcePolicy.containerPolicies.containerName**

строка

Имя контейнера или `DefaultContainerResourcePolicy`, в этом случае политика применяется к контейнерам, для которых не указана собственная политика.

- **spec.resourcePolicy.containerPolicies.controlledResources**

массив строк

Указывает тип рекомендаций, которые будут вычисляться (и, возможно, применяться) VPA. Если не указано, используется значение по умолчанию: `ResourceCPU`, `ResourceMemory`.

По умолчанию: `["cpu", "memory"]`

- **Элемент массива**

строка

Имя, которое идентифицирует различные ресурсы в списке ресурсов (`ResourceList`).

- **spec.resourcePolicy.containerPolicies.controlledValues**

строка

Указывает, какие значения ресурсов должны контролироваться.

По умолчанию: `RequestsAndLimits`

Допустимые значения: `RequestsAndLimits`, `RequestsOnly`

- **spec.resourcePolicy.containerPolicies.maxAllowed**

объект

Указывает максимальное количество ресурсов, которое будет рекомендовано для контейнера. По умолчанию максимальное значение не установлено.

- **spec.resourcePolicy.containerPolicies.minAllowed**

объект

Указывает минимальное количество ресурсов, которое будет рекомендовано для контейнера. По умолчанию минимальное значение не установлено.

- **spec.resourcePolicy.containerPolicies.mode**

строка

Указывает, включен ли автоскейлер для контейнера.

По умолчанию: `Auto`

Допустимые значения: `Auto`, `Off`

- **spec.scope**

строка

Включает `scoped`-режим рекомендаций для `targetRef.kind=DaemonSet`. Значение должно быть ключом лейбла узла (например, `node.kubernetes.io/instance-type`).

В `scoped`-режиме рекомендации группируются по `status.groups[].scopeValue`. Для остальной нагрузки и для `DaemonSet` без `scope` используется обычное поле `status.recommendation`.

- **spec.targetRef**

объект

Обязательный параметр

Указывает на контроллер, управляющий набором подов, которые должен контролировать автоскейлер, например, Deployment или StatefulSet.

VerticalPodAutoscaler может быть нацелен на контроллер, который реализует подресурс масштабирования (набор подов извлекается из ScaleStatus контроллера) или на другие известные контроллеры (например, для DaemonSet набор подов читается из спецификации контроллера).

Если VerticalPodAutoscaler не может использовать указанный объект, он сообщит о состоянии ConfigUnsupported. Обратите внимание, что VerticalPodAutoscaler не требует полной реализации подресурса масштабирования — он не будет использовать его для изменения количества реплик. Единственное, что используется — это селектор лейблов, соответствующих подам, сгруппированным по целевому ресурсу.

- **spec.targetRef.apiVersion**

строка

Версия API связанного объекта.

- **spec.targetRef.kind**

строка

Обязательный параметр

Тип связанного объекта.

Больше информации в документации.

- **spec.targetRef.name**

строка

Обязательный параметр

Имя связанного объекта.

Больше информации в документации.

- **spec.updatePolicy**

объект

Описывает правила применения изменений к подам. Если не указано, все поля в PodUpdatePolicy устанавливаются в значения по умолчанию.

- **spec.updatePolicy.evictionRequirements**

массив объектов

Список требований для завершения работы пода (EvictionRequirements), которые должны быть выполнены, чтобы под завершил работу (Eviction). Если указано несколько требований, все они должны быть выполнены для разрешения завершения работы.

Определяет одно условие, которое должно быть истинным для того, чтобы под завершил работу.

- **spec.updatePolicy.evictionRequirements.changeRequirement**

строка

Обязательный параметр

Описывает зависимость между новым целевым рекомендуемым значением для пода и его текущими запросами, а также тип изменения, необходимого для того, чтобы под завершил работу.

Допустимые значения: TargetHigherThanRequests, TargetLowerThanRequests

- **spec.updatePolicy.evictionRequirements.resources**

массив строк

Обязательный параметр

Это список одного или нескольких ресурсов, к которым применяется условие. Если указано несколько ресурсов, условие для завершения работы

считается выполненным, если хотя бы один из ресурсов удовлетворяет `changeRequirement`.

- **Элемент массива**
строка

Это имя, идентифицирующее различные ресурсы в списке ресурсов (`ResourceList`).

- **`spec.updatePolicy.minReplicas`**
целочисленный

Минимальное количество реплик, которые должны быть в живом состоянии, чтобы обновляющий процесс (`Updater`) попытался удалить под (с учётом других проверок, таких как `PDB`). Разрешены только положительные значения. Перезаписывает глобальный флаг `--min-replicas`.

- **`spec.updatePolicy.updateMode`**
строка

Режим работы контроллера VPA.

- `Auto` — режимы `Auto` и `Recreate` выполняют одинаковую функцию. Этот режим зарезервирован для обновления ресурсов пода в Kubernetes.
- `Recreate` — позволяет VPA изменять требования к ресурсам работающих подов (т.е. перезапускать их в процессе работы). Этот режим может привести к временной недоступности сервиса (из-за перезапуска), если у вас только одна реплика (`replicas: 1`). В этом режиме VPA не пересоздаёт поды, которые не были созданы контроллером.
- `Initial` — VPA изменяет ресурсы подов только при их запуске (но не во время работы подов).
- `Off` — VPA не предпринимает никаких действий для обновления запросов ресурсов для работающих контейнеров. Автоскейлер рассчитывает рекомендации и сохраняет их в поле статуса объекта VPA. Вы можете просмотреть рекомендации VPA с помощью следующей команды: `d8 k describe vpa <vpa-name>`.
- `InPlaceOrRecreate (GA)` — работает в версиях Kubernetes 1.33 и старше. VPA попытается изменить ресурсы пода без перезагрузки и при невозможности пересоздаст его.

По умолчанию: `InPlaceOrRecreate`

Допустимые значения: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **`status`**
объект

Текущая информация об автоскейлере.

- **`status.conditions`**
массив объектов

Набор условий, необходимых для того, чтобы автоскейлер мог масштабировать свою цель, и указывает, выполнены ли эти условия.

Описывает состояние `VerticalPodAutoscaler` в определённый момент времени.

- **`status.conditions.lastTransitionTime`**
строка

Время последнего изменения состояния условия с одного статуса на другой.

- **`status.conditions.message`**
строка

Сообщение, которое содержит объяснение и детали о переходе.

- **`status.conditions.reason`**
строка

Причина последнего перехода состояния.

- **`status.conditions.status`**
строка

Обязательный параметр

Состояние условия (`True` , `False` , `Unknown`).

- **status.conditions.type**

строка

Обязательный параметр

Описывает текущее состояние условия.

- **status.groups**

массив объектов

Сгруппированные рекомендации по значениям `spec.scope` для `scoped DaemonSet`.

В этом режиме `status.groups` является источником истины, а `status.recommendation` не заполняется.

- **status.groups.containerRecommendations**

массив объектов

Рекомендации по контейнерам для группы.

В `scoped`-режиме поле хранится в компактном виде (в первую очередь `containerName` и `target`).

- **status.groups.containerRecommendations.containerName**

строка

Имя контейнера.

- **status.groups.containerRecommendations.lowerBound**

объект

Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

- **status.groups.containerRecommendations.scope**

строка

Необязательное поле совместимости.

Для `scoped DaemonSet` фактическая область рекомендации определяется через `spec.scope` + `scopeValue` группы.

- **status.groups.containerRecommendations.target**

объект

Обязательный параметр

Recommended amount of resources. Observes `ContainerResourcePolicy`.

- **status.groups.containerRecommendations.uncappedTarget**

объект

The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`.

May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than `MinAllowed` or higher than `MaxAllowed`). Used only as status indication, will not affect actual resource assignment.

- **status.groups.containerRecommendations.upperBound**

объект

Maximum recommended amount of resources.

Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

- **status.groups.scopeValue**

строка

Обязательный параметр

Значение лейбла `spec.scope` узла, для которого рассчитана группа.

Специальное значение `__absent__` означает, что у узла отсутствует ключ лейбла `spec.scope`.

- **status.recommendation**
объект

Последнее вычисленное количество ресурсов, рекомендованных автоскейлером для контролируемых подов.

Используется для обычного режима (Deployment/StatefulSet/DaemonSet без `spec.scope`).

- **status.recommendation.containerRecommendations**
массив объектов

Ресурсы, рекомендованные автоскейлером для каждого контейнера.

Рекомендации по ресурсам, вычисленные автоскейлером для конкретного контейнера. Учитывает политику ресурсов контейнера, если она указана в спецификации. В частности, рекомендация не генерируется для контейнеров, у которых установлен режим `ContainerScalingMode` равный 'Off'.

- **status.recommendation.containerRecommendations.containerName**
строка

Имя контейнера.

- **status.recommendation.containerRecommendations.lowerBound**
объект

Минимальное рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`. Это количество ресурсов не гарантирует, что приложение будет работать стабильно, однако использование меньшего количества ресурсов, скорее всего, окажет значительное влияние на производительность и доступность.

- **status.recommendation.containerRecommendations.scope**
строка

Связка `<ключ scope>=<значение>`.

- **status.recommendation.containerRecommendations.target**
объект

Обязательный параметр

Рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`.

- **status.recommendation.containerRecommendations.uncappedTarget**
объект

Наиболее ранняя рекомендованная цель ресурсов, вычисленная автоскейлером для контролируемых подов, основанная только на фактическом использовании ресурсов, без учета `ContainerResourcePolicy`. Может отличаться от рекомендации, если фактическое использование ресурсов приводит к нарушению `ContainerResourcePolicy` (меньше чем `MinAllowed` или больше чем `MaxAllowed`). Используется только как индикатор состояния, не влияет на фактическое назначение ресурсов.

- **status.recommendation.containerRecommendations.upperBound**
объект

Максимально рекомендованное количество ресурсов. Учитывает `ContainerResourcePolicy`. Ресурсы, выделенные сверх этого значения, скорее всего, будут потрачены впустую. Это значение может быть больше, чем максимальное количество ресурсов, которое приложение фактически способно потреблять.

VerticalPodAutoscalerCheckpoint

Short names: `vpacheckpoint`

Scope: Namespaced

- v1
- v1beta2

Чекпоинт внутреннего состояния VPA, используемый для восстановления после перезапуска механизма рекомендаций.

- **apiVersion**
строка

Определяет версионированную схему представления этого объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации.

- **kind**
строка

Значение, представляющее REST-ресурс, который описывает данный объект. Серверы могут определить его из конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации.

- **metadata**
объект
- **spec**
объект

Спецификация чекпоинта.

Больше информации в документации.

- **spec.containerName**
строка

Название контейнера, для которого был создан чекпоинт.

- **spec.vpaObjectName**
строка

Название объекта VPA, в котором хранится объект VerticalPodAutoscalerCheckpoint.

- **status**
объект

Data of the checkpoint.

- **status.cpuHistogram**
объект

Checkpoint of histogram for consumption of CPU.

- **status.cpuHistogram.bucketWeights**
объект

Map from bucket index to bucket weight.

- **status.cpuHistogram.referenceTimestamp**
строка

Reference timestamp for samples collected within this histogram.

- **status.cpuHistogram.totalWeight**
число

Sum of samples to be used as denominator for weights from BucketWeights.

- **status.firstSampleStart**
строка

Timestamp of the first sample from the histograms.

- **status.lastSampleStart**
строка

Timestamp of the last sample from the histograms.

- **status.lastUpdateTime**

строка

The time when the status was last refreshed.

- **status.memoryHistogram**

объект

Checkpoint of histogram for consumption of memory.

- **status.memoryHistogram.bucketWeights**

объект

Map from bucket index to bucket weight.

- **status.memoryHistogram.referenceTimestamp**

строка

Reference timestamp for samples collected within this histogram.

- **status.memoryHistogram.totalWeight**

число

Sum of samples to be used as denominator for weights from BucketWeights.

- **status.totalSamplesCount**

целочисленный

Total number of samples in the histograms.

- **status.version**

строка

Version of the format of the stored data.

Устаревший ресурс. Поддержка ресурса может быть исключена в следующих версиях.

Чекпоинт внутреннего состояния VPA, используемый для восстановления после перезапуска механизма рекомендаций.

- **apiVersion**

строка

Определяет версионированную схему представления этого объекта. Серверы должны преобразовывать распознанные схемы в последнее внутреннее значение и могут отклонять нераспознанные значения.

Больше информации в документации.

- **kind**

строка

Значение, представляющее REST-ресурс, который описывает данный объект. Серверы могут определить его из конечной точки, к которой клиент отправляет запросы. Не может быть изменено. Используется в CamelCase.

Больше информации в документации.

- **metadata**

объект

- **spec**

объект

Спецификация чекпоинта.

Больше информации в документации.

- **spec.containerName**

строка

Название контейнера, для которого был создан чекпоинт.

- **spec.vpaObjectName**

строка

Название объекта VPA, в котором хранится объект VerticalPodAutoscalerCheckpoint.

- **status**

объект

Data of the checkpoint

data of the checkpoint.

- **status.cpuHistogram**

объект

Checkpoint of histogram for consumption of CPU.

- **status.cpuHistogram.bucketWeights**

объект

Map from bucket index to bucket weight.

- **status.cpuHistogram.referenceTimestamp**

строка

Reference timestamp for samples collected within this histogram.

- **status.cpuHistogram.totalWeight**

число

Sum of samples to be used as denominator for weights from BucketWeights.

- **status.firstSampleStart**

строка

Timestamp of the first sample from the histograms.

- **status.lastSampleStart**

строка

Timestamp of the last sample from the histograms.

- **status.lastUpdateTime**

строка

The time when the status was last refreshed.

- **status.memoryHistogram**

объект

Checkpoint of histogram for consumption of memory.

- **status.memoryHistogram.bucketWeights**

объект

Map from bucket index to bucket weight.

- **status.memoryHistogram.referenceTimestamp**

строка

Reference timestamp for samples collected within this histogram.

- **status.memoryHistogram.totalWeight**

число

Sum of samples to be used as denominator for weights from BucketWeights.

- **status.totalSamplesCount**

целочисленный

Total number of samples in the histograms.

- **status.version**

строка

Version of the format of the stored data.

FAQ

Как посмотреть рекомендации Vertical Pod Autoscaler?

После создания кастомного ресурса VerticalPodAutoscaler посмотреть рекомендации VPA можно следующим образом:

```
d8 k describe vpa my-app-vpa
```

В секции `status` отобразятся параметры:

- **Target** — количество ресурсов, которое будет оптимальным для пода (в пределах resourcePolicy);
- **Lower Bound** — минимальное рекомендуемое количество ресурсов для более или менее (но не гарантированно) хорошей работы приложения;
- **Upper Bound** — максимальное рекомендуемое количество ресурсов. Скорее всего, ресурсы, выделенные сверх этого значения, идут в мусорку и совсем никогда не нужны приложению;
- **Uncapped Target** — рекомендуемое количество ресурсов в самый последний момент, то есть данное значение считается на основе самых крайних метрик, не смотря на историю ресурсов за весь период.

Как Vertical Pod Autoscaler работает с лимитами?

Пример 1

В примере представлен VPA-объект:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"
```

В VPA-объекте представлен под с ресурсами:

```
resources:
  limits:
    cpu: 2
  requests:
    cpu: 1
```

Если контейнер использует весь CPU, и VPA рекомендует этому контейнеру 1.168 CPU, то отношение между запросами и ограничениями будет равно 100%. В этом случае при пересоздании пода VPA модифицирует его и проставит такие ресурсы:

```
resources:
  limits:
    cpu: 2336m
  requests:
    cpu: 1168m
```

Пример 2

В примере представлен VPA-объект:


```

---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"

```

В VPA-объекте представлен под с ресурсами:

```

resources:
  limits:
    cpu: 1
  requests:
    cpu: 750m

```

Если отношение запросов и ограничений равно 25%, и VPA рекомендует 1.168 CPU для контейнера, VPA изменит ресурсы контейнера следующим образом:

```

resources:
  limits:
    cpu: 1557m
  requests:
    cpu: 1168m

```

Если необходимо ограничить максимальное количество ресурсов, которые могут быть выделены для ограничений контейнера, нужно использовать в спецификации объекта VPA `maxAllowed` или использовать Limit Range объекта Kubernetes.

Примеры

Настройка модуля

```

apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system

```

Пример минимального custom resource VerticalPodAutoscaler

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-app
```

Пример полного custom resource VerticalPodAutoscaler

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "InPlaceOrRecreate"
  resourcePolicy:
    containerPolicies:
      - containerName: hamster
        minAllowed:
          memory: 100Mi
          cpu: 120m
        maxAllowed:
          memory: 300Mi
          cpu: 350m
    mode: Auto
```